Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Plataforma web para la ejecución de modelos de clasificación como servicio

Estudiante: Jorge Juan Gabín Brenlla
Dirección: Javier Parapar López
Dirección: Alfonso Landín Piñeiro

A Coruña, setembro de 2020.

A Mamá, Papá y Dani

Agradecimientos

En primer lugar, quiero dar las gracias a mis dos tutores del Trabajo de Fin de Grado, Javi y Alfonso, por su gran dedicación y su gran apoyo tanto personal como profesionalmente. Repasando estos meses en los cuales he trabajado con ellos me siento afortunado de que hayan confiado en mi para llevar a cabo este proyecto juntos.

A toda mi familia, que siempre ha confiado en mi y me ha apoyado a lo largo de todo este tiempo.

Y por último a todos los compañeros con los que he trabajado a lo largo de toda la carrera, en especial a Marcos, Jaime, Bieito y Miguel, con los que he compartido grandes momentos a lo largo de esta etapa.

Gracias.

Jorge Juan Gabín Brenlla
A Coruña, 6 de septiembre de 2020

Resumen

El exponencial crecimiento del volumen de datos almacenados en la nube en los últimos años, junto a la incipiente necesidad de analizarlos con el fin de obtener conocimiento a partir de los mismos, ha provocado la llamada revolución de los datos.

En este contexto nace **Modelab**, una plataforma web para la ejecución de modelos de clasificación como servicio.

La finalidad de esta plataforma es ofrecer a los usuarios una forma sencilla y rápida de realizar clasificaciones sobre grandes conjuntos de datos almacenados en la misma y ofrecer los resultados de estas clasificaciones de forma clara y concisa.

Abstract

The exponential growth of the volume of data stored in the cloud the last few years, together with the incipient need to analyse it in order to obtain knowledge from it, has led to the so-called data revolution.

In this context, **Modelab**, a web platform for the execution of classification models as a service, was born.

The purpose of this platform is to offer users a simple and fast way of carrying out classifications on large datasets stored on it and to offer the results of these classifications in a clear and concise way.

Palabras clave:

- Aprendizaje máquina
- Clasificación
- Ciencia de datos
- SCRUM
- Java
- Python
- JavaScript
- Spring
- Scikit-learn
- ReactJs

Keywords:

- Machine learning
- Classification
- Data Science
- SCRUM
- Java
- Python
- JavaScript
- Spring
- Scikit-learn
- ReactJs

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo comentaremos la motivación, los objetivos y el plan de trabajo seguido en el proyecto. Además, se explicará la estructura que seguirá la memoria.

1.1 Motivación

El volumen de información global crece día a día de forma exponencial. Como podemos ver en la *Figura 1.1*, el volumen de datos esperado para el año 2025, según la empresa de inteligencia de negocio, International Data Corporation (IDC), será de 175 zettabytes, multiplicando por 175 el volumen del año 2011.

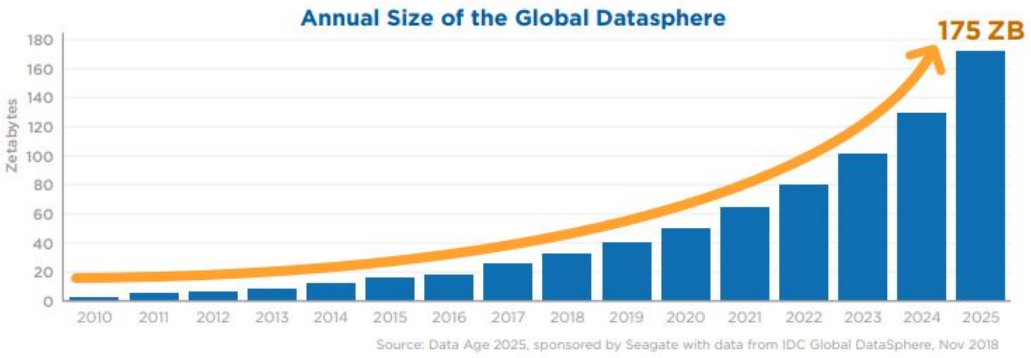

Figura 1.1: Crecimiento anual del volumen de datos

Sin embargo, la llamada revolución de los datos no se debe simplemente al crecimiento de la cantidad de datos disponibles. Fundamentalmente, se debe a las tecnologías que a día de hoy se usan para extraer, almacenar, analizar y transformar estos datos. Estos cuatro aspectos permiten obtener mucho conocimiento oculto en las profundidades de los datos, y de esta forma adquirir nuevos conocimientos, descubrir nuevas conexiones entre los datos y realizar nuevas predicciones.

Las nuevas herramientas software nos dan la posibilidad de usar conjuntos de datos para tomar mejores decisiones, basadas en hechos y no en instintos, intuiciones o conjeturas. Estas nuevas herramientas, incluyendo los sistemas de aprendizaje máquina y las tecnologías de modelado y simulación, están ayudando a dar un propósito a los datos, transformándolos de manera que pueden ayudarnos a extrapolar, visualizar, refinar, modelar y predecir.

La motivación de este proyecto es crear una plataforma que forme parte de este conjunto de herramientas, proveyendo a los usuarios una interfaz sencilla a través de la cual ejecutar, en primera instancia, una serie de algoritmos de clasificación [1] sobre los conjuntos de datos subidos a la plataforma. De esta forma, tanto usuarios experimentados como aquellos que se inician en el mundo del *machine learning* [2] pueden tener la posibilidad de entrenar modelos y clasificar sus datos, pudiendo más tarde hacer uso de estos modelos para realizar predicciones sobre nuevos *datasets*.

1.2 Objetivos

Los principales objetivos que se pretenden alcanzar en el desarrollo del proyecto son los siguientes:

- **API de Machine Learning:** Construcción de un Application Programming Interface (API) que permita gestionar los *datasets* subidos por los usuarios y gestionar la ejecución de algoritmos de clasificación sobre ellos.
- **API de Operaciones Básicas:** Implementación de un API que permita gestionar usuarios y realizar búsquedas sobre los *datasets* y las ejecuciones almacenadas en la plataforma.
- **Aplicación frontend:** Diseño e implementación de un frontal que permita a los usuarios interactuar con los APIs desarrollados.
- **Sistema de monitorización:** Construcción de un sistema de monitorización que permita conocer el estado del sistema en todo momento.

1.3 Estructura de la memoria

La memoria seguirá la siguiente estructura:

- **Introducción:** Expone el contexto en el que se desarrollará el proyecto, y se tratan los aspectos relacionados con el objetivo y el alcance del mismo. Además se detalla la estructura del presente documento, así como el plan de trabajo que se seguirá para el desarrollo del proyecto.

- **Conceptos previos:** Detalla conceptos relacionados con el *machine learning*, más concretamente, con los algoritmos de clasificación y métodos de extracción de características, cuyo conocimiento es necesario para un mejor entendimiento del proyecto.
- **Tecnología:** Expone y justifica las tecnologías usadas para el desarrollo del proyecto.
- **Proceso de ingeniería:** Trata todos los aspectos relacionados con el proceso de ingeniería seguido en el proyecto: metodología, planificación y gestión del proyecto.
- **Desarrollo:** Detalla el análisis, diseño, implementación y pruebas de cada uno de los componentes del sistema.
- **Conclusiones:** Expone el estado final del proyecto, el posible trabajo futuro a realizar sobre el mismo y la relación del desarrollo del mismo con las competencias adquiridas en el grado.
- **Apéndice:** Está compuesto por dos glosarios que recogen los términos y acrónimos usados en el proyecto:
  - **Glosario de términos**
  - **Glosario de acrónimos**
- **Bibliografía:** Recoge la documentación sobre la que el alumno se ha apoyado para el desarrollo del proyecto.

1.4 Plan de trabajo

Desde un punto de vista general se han seguido los siguientes puntos para llevar a cabo el desarrollo del proyecto:

- Lectura y estudio de documentación acerca de los algoritmos de clasificación y herramientas para su desarrollo.
- Lectura y estudio de documentación acerca del tratamiento de *datasets* y herramientas para llevar a cabo dicha tarea.
- Selección de los algoritmos de clasificación que proveerá la plataforma.
- Desarrollo de los distintos componentes de la plataforma.
- Redacción de la memoria, de forma paralela al desarrollo del resto de puntos.

Capítulo 2

Conceptos previos

En este capítulo trataremos conceptos referentes a los Modelos de Clasificación. Además, se detallan los distintos modelos que ofrecerá la plataforma que se ha construido como Trabajo de Fin de Grado.

2.1 Aprendizaje supervisado

El aprendizaje supervisado son un conjunto de técnicas que permiten realizar predicciones futuras basadas en comportamientos o características analizadas en datos históricos etiquetados.

La clasificación [1] es una subcategoría del aprendizaje supervisado en la que el objetivo es predecir las etiquetas de clase categóricas (discreta, valores no ordenados, pertenencia a grupo) de las nuevas instancias, basándonos en observaciones pasadas.

Este tipo de aprendizaje supervisado es una de las tareas llevada a cabo de forma más frecuente por los denominados Sistemas Inteligentes. Por ello, un gran número de paradigmas desarrollados o bien por la Estadística o bien por la Inteligencia Artificial, son capaces de realizar las tareas propias de la clasificación.

Dentro de los algoritmos de clasificación existen dos tipos principales: los *lazy learners* y los *eager learners*.

Los *lazy learners* simplemente almacenan los datos de entrenamiento y esperan a que lleguen los datos de prueba. Una vez llegan estos datos la clasificación se realiza según las relaciones entre los datos de entrenamiento y test. Por esto, en estos algoritmos la etapa de entrenamiento es rápida y la de predicción lenta. Un ejemplo es K-Nearest Neighbors (K-NN).

Por otro lado, los *eager learners* construyen un modelo de clasificación basado en los datos de entrenamiento dados antes de recibir los datos para la clasificación. Estos algoritmos deben ser capaces de encontrar una única hipótesis sobre la cual se clasificarán el resto de datos. Debido a la construcción del modelo, estos toman mucho tiempo para entrenar y menos tiempo

para predecir. Un ejemplo de de estos algoritmos son los árboles de decisión.

Como paso previo a la aplicación de un método de clasificación es necesario llevar a cabo la partición del conjunto de datos en dos subconjuntos más pequeños. Uno de estos subconjuntos se usará para llevar a cabo el entrenamiento del modelo, mientras que el otro se usará para testearlo. El subconjunto de entrenamiento se utiliza para estimar los parámetros del modelo, y el de test para comprobar el comportamiento del modelo estimado. Lo ideal es entrenar el modelo con un conjunto de datos independiente de los datos con los que realizaremos el test.

2.2 Algoritmos de clasificación

A continuación, trataremos de forma individual cada uno de los modelos de clasificación que tras haber sido estudiados se han elegido para formar parte del sistema.

2.2.1 Logistic Regression

En estadística, la regresión logística [3] es un tipo de análisis de regresión usado para predecir el resultado de una variable categórica en función de una serie de variables independientes o predictoras.

La regresión logística unidimensional permite correlacionar la probabilidad de una variable cualitativa binaria con una variable escalar. La idea es que este método aproxime cual es la probabilidad de obtener "0" (no ocurre cierto suceso) o "1" (ocurre cierto suceso) con el valor de la variable explicativa x.

La regresión logística es, por lo tanto, un caso especial de un análisis de regresión en el cual la variable dependiente es dicotómica ("Sí" [1] o"No" [0]).

Figura 2.1: Logistic Regression

2.2.2 K-Nearest Neighbors

El método K-Nearest Neighbors (K-NN) o K-Vecinos más cercanos [4] es un método de clasificación que sirve para estimar la función de densidad $F(x/C_j)$ de las predictoras x por cada clase C_j.

Es un método de clasificación no paramétrico que estima la probabilidad o incluso la función de densidad de probabilidad (probabilidad relativa de que una variable tome un determinado valor) de que un elemento x pertenezca a la clase C_j.

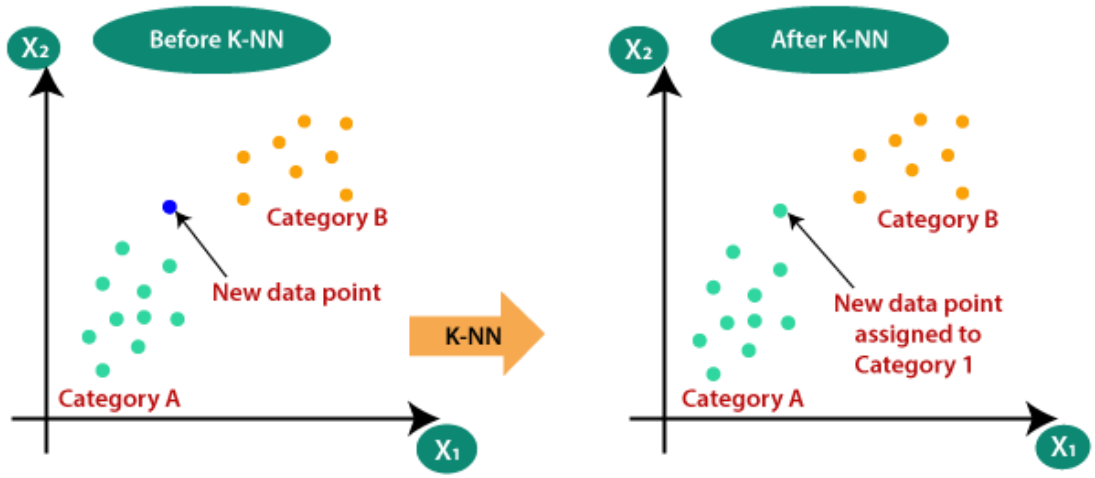

Figura 2.2: K-Nearest Neighbors

2.2.3 Naïve Bayes

Un clasificador Naïve Bayes [5] es un clasificador probabilístico fundamentado en el teorema de Bayes junto con algunas hipótesis simplificadoras adicionales. Son estas hipótesis, que determinan la independencia de las variables predictoras, las que le atribuyen el apelativo de *naïve* o ingenuo.

Este clasificador asume que la ausencia o presencia de una característica no está relacionada con la ausencia o presencia de cualquier otra característica. Por ello, un clasificador ingenuo de Bayes considera que cada una de las características de un elemento contribuye de manera independiente a la probabilidad de que ese elemento forme parte de un conjunto o no, independientemente del resto de características.

Una gran ventaja de este método es que requiere una pequeña cantidad de datos de entrenamiento para estimar los parámetros necesarios para la clasificación.

El teorema de Bayes se expresa mediante la siguiente ecuación:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \tag{2.1}$$

Lo que quiere expresar esta ecuación es que la probabilidad de que se dé H habiéndose dado D es igual a la probabilidad de que se dé D habiéndose dado H multiplicada por la probabilidad de H y dividida por la probabilidad de D.

2.2.4 Stochastic Gradient Descent

Antes de comenzar a explicar el gradiente descendente estocástico es importante comentar en qué consiste el algoritmo del gradiente descendente. En primer lugar, un gradiente es la pendiente de una función. Este algoritmo por tanto trata de medir el cambio de una variable tras realizar cambios en otra variable. Partiendo de un valor inicial, el algoritmo se ejecuta iterativamente para encontrar los valores óptimos de los distintos parámetros para finalmente encontrar el mínimo valor para una determinada función de coste.

El Stochastic Gradient Descent (SGD) [6] surge como una alternativa al gradiente descendente que busca disminuir el alto coste computacional de este. Para ello, como indica la propia palabra estocástico, este algoritmo selecciona un conjunto reducido de muestras al azar del conjunto de datos sobre los cuales operar en cada iteración en lugar de operar sobre todo el conjunto cada iteración. A pesar de que usar todo el conjunto de datos es realmente útil para conseguir el mínimo en la función de costa de forma más precisa, existen situaciones en las que debido al tamaño de los conjuntos de datos esta solución es totalmente inviable.

Estrictamente hablando, el SGD es meramente una técnica de optimización y no corresponde a una familia específica de modelos de *machine learning*. Es sólo una forma de entrenar un modelo. Por ello, y para que este funcione como un clasificador en la plataforma, el usuario podrá escoger la función de pérdida a partir de la cual se construirá el modelo el cual será entrenado mediante SGD.

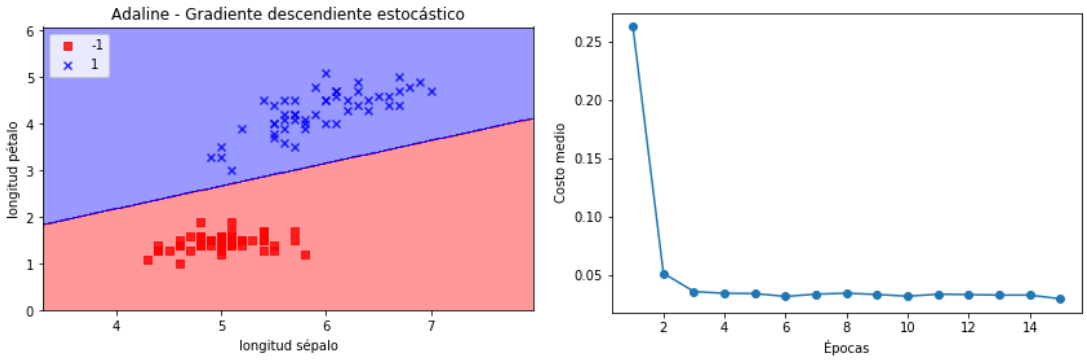

Figura 2.3: Stochastic Gradient Descent

2.2.5 Decision Tree

Los árboles de decisión [7] no son un modelo basado en la estimación de los parámetros de la ecuación, por lo que no existe la necesidad de estimar un modelo estadístico formal. Este tipo de algoritmo se basan en las participaciones sucesivas para llevar a cabo la clasificación. Son muy útiles cuando tratamos un número muy elevado de datos, ya que permiten revelar formas complejas en la estructura de dichos datos no detectables mediante otros métodos convencionales de regresión.

Los árboles de decisión cuentan con tres componentes principales: los nodos, las ramas y las hojas. Los nodos representan las variables de entrada, las ramas los posibles valores que pueden tomar estas variables y las hojas representan los posibles valores de salida.

Este algoritmo aprende a realizar particiones en base al valor de las variables, llevando a cabo una partición del árbol de forma recursiva. Los árboles de decisión tienen la ventaja de que son muy fáciles de entender e interpretar, ya que su visualización es muy similar a un diagrama de flujo el cual imita fácilmente el pensamiento a nivel humano.

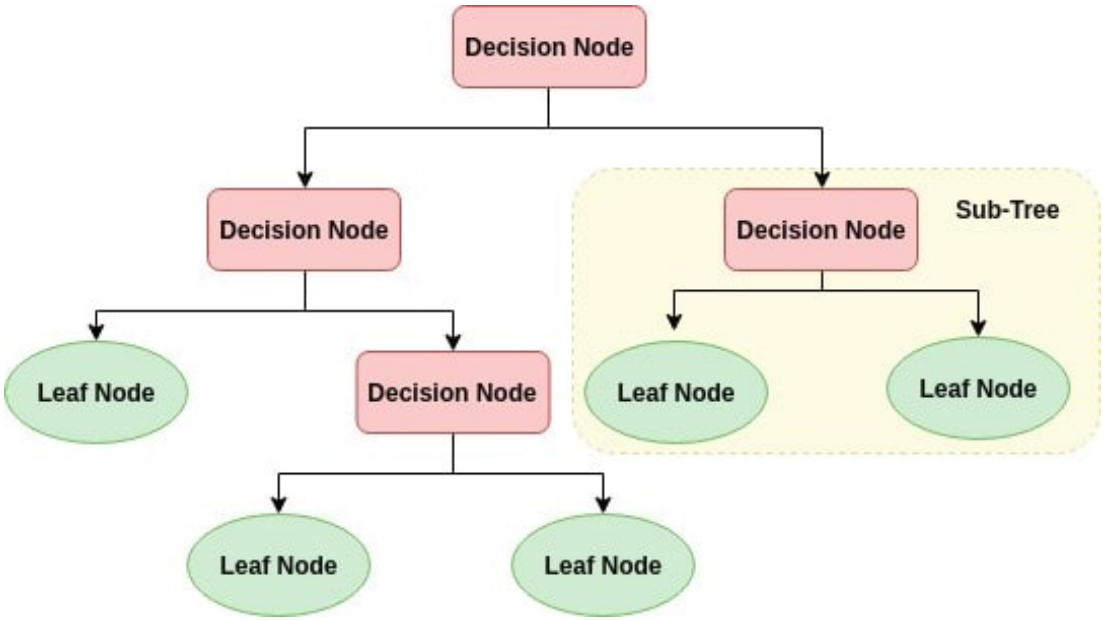

Figura 2.4: Decision Tree

2.2.6 Random Forest

Antes de comenzar con la explicación de este algoritmo es necesario conocer el concepto de *bagging*.

El *bagging* (*bootstrap aggregation*) es una técnica que se usa con el fin de reducir la varianza de las predicciones mediante la combinación de los resultados de distintos clasificadores modelados con subconjuntos de datos distintos de una misma población.

El Random Forest [7] [8] es un modificación de esta técnica que consigue mejores resultados eliminando la correlación entre los árboles que se generan durante el proceso. Este algoritmo es muy útil tanto en problemas de clasificación como en problemas de regresión. Construyes un grupo de modelos "débiles" los cuales formarán un modelo robusto tras combinarse.

Este algoritmo genera por tanto una serie de árboles, cada uno de ellos dará un resultado para la clasificación (cada árbol vota por una determinada clase). finalmente, la clase más votada será la escogida como resultado para la clasificación.

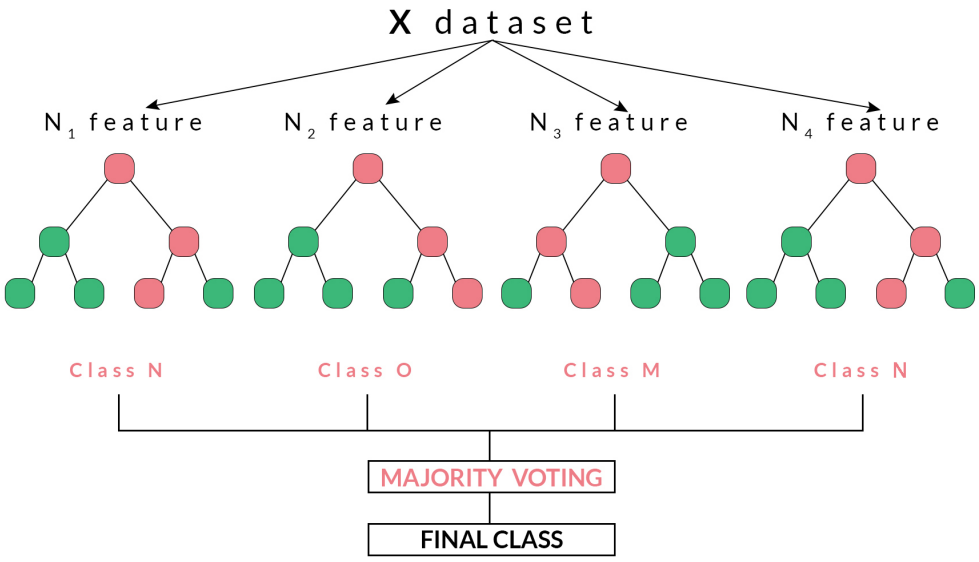

Figura 2.5: Random Forest

2.2.7 Support Vector Machine

Las Máquinas de Soporte Vectorial [9] son un conjunto de algoritmos de aprendizaje supervisado que desarrollan métodos relacionados con los problemas de clasificación y regresión.

La Support Vector Machine (SVM), etiqueta los datos de entrenamiento en diferentes clases y los representa como puntos en el espacio, con el fin de tratar de separar las clases mediante el mayor espacio posible. Tras realizar dicha separación en la fase de entrenamiento, los datos de test son clasificados en función a su proximidad con cada clase.

La característica fundamental de estos algoritmos reside en el concepto de **separación óptima**. Las SVM buscan encontrar el hiper-plano que tenga la mayor distancia con los puntos que estén más cerca del mismo. De esta forma, los puntos del vector que están a un lado de este hiper-plano se etiquetan con una determinada categoría y los que se encuentran al otro lado con otra.

La manera más sencilla de llevar a cabo la separación es mediante una línea recta o un plano recto, como podemos ver en la *Figura 2.6* de ejemplo. Sin embargo, en los conjuntos de datos a clasificar no se suele dar el caso ideal de que solo existan dos dimensiones, si no que existirán más de dos variables predictoras, curvas no lineales de separación, clasificaciones en más de dos categorías, etc.

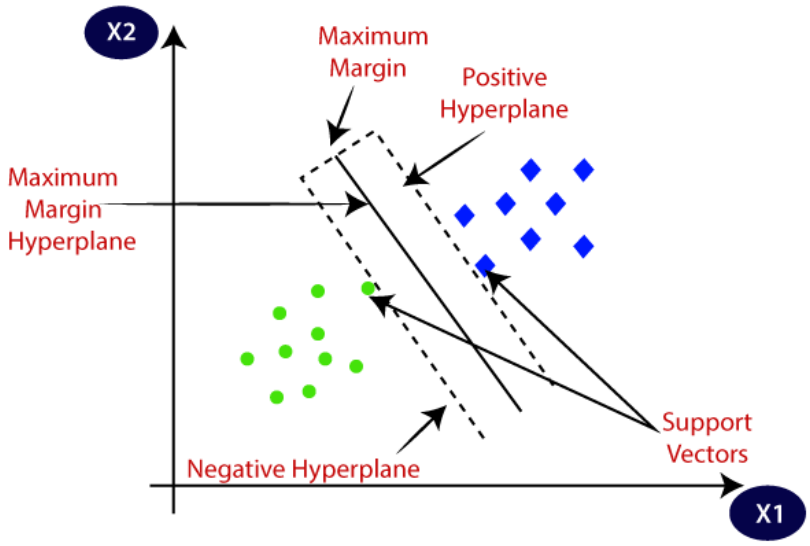

Figura 2.6: Support Vector Machine

2.3 Extracción de caracterísitcas

La extracción de *features* o extracción de características [10] de un *dataset* es un proceso cuyo objetivo es reducir la dimensionalidad del conjunto de datos. Los grandes conjuntos de datos necesitan muchos recursos para ser procesados, y por ello nace este proceso. La extracción de características es el nombre de los métodos que seleccionan y/o combinan variables en características, reduciendo efectivamente la cantidad de datos que deben procesarse, sin dejar de describir con precisión y por completo el conjunto de datos original.

Este proceso es extremadamente útil cuando es necesario reducir el número de recursos necesarios para el procesamiento sin llegar a perder información importante o relevante. Por estas razones, con el fin de poder desplegar la plataforma en un entorno con recursos limitados, se ofrecen una serie de métodos de extracción de características que se pueden ejecutar de forma previa a las clasificaciones.

Capítulo 3
Tecnología

En el presente capítulo se tratan y justifican las principales decisiones con respecto a las tecnologías utilizadas para el desarrollo del proyecto. La elección de dichas tecnologías es un factor altamente relevante para el correcto desarrollo del mismo, así como para finalizarlo de manera exitosa.

Las cuestiones tecnológicas del proyecto se pueden dividir en tres grandes bloques: el API de Machine Learning, el API de Operaciones Básicas y la aplicación frontal de la plataforma. Además, a parte de estos tres bloques trataremos los aspectos referentes a las tecnologías utilizadas para la construcción del sistema de *logs*, análisis y monitorización, la gestión de la persistencia y las herramientas necesarias para dar soporte al desarrollo.

3.1 API de Machine Learning

El primer módulo del que hablaremos es el encargado de ofrecer los distintos modelos de clasificación del sistema y de realizar la gestión de los datasets de los usuarios. En este apartado se definirán por lo tanto las tecnologías usadas para la ejecución de los modelos de clasificación y el almacenamiento de los resultados, el análisis y almacenamiento de los datasets subidos por los usuarios y la implementación de un API REST que recibirá peticiones directamente desde el frontal para obtener los mejores tiempos de respuesta posibles.

3.1.1 Lenguaje de programación: Python

Python[1] [11] es un lenguaje de programación multiparadigma, que incorpora soporte para programación orientada a objetos, programación imperativa y en menor medida programación funcional. Es un lenguaje interpretado que usa tipado dinámico.

La elección de Python para este módulo del sistema se debe principalmente a la elección de la librería Scikit-learn para la ejecución de los modelos de clasificación y Pandas para la

[1] https://www.python.org

gestión y análisis de los datasets. Ambas librerías son exclusivas de Python, y aportan lo que se necesita para el desarrollo de una plataforma de las características como la que aquí se trata.

3.1.2 Librería para la ejecución de los modelos de clasificación: Scikit-learn

Scikit-learn[2] [2] [12] es una biblioteca libre para aprendizaje automático en Python. Esta biblioteca incluye una gran variedad de algoritmos de clasificación, regresión y análisis de grupos.

La mayor ventaja que ofrece sklearn con respecto a otras librerías de *machine learning* como por ejemplo, TensorFlow[3] o PyTorch[4], es su gran accesibilidad y simplicidad, es decir, cuenta con una gran documentación y, además, su curva de aprendizaje se ajusta a lo que se buscaba para el presente proyecto. En cuanto al rendimiento, es cierto que existen otras librerías mejores, sin embargo sklearn se comporta perfectamente en este aspecto para los requerimientos del sistema.

3.1.3 Librería para la gestión de los datasets: Pandas

Pandas[5] es una extensión de Numpy[6] para la manipulación y análisis de datos para el lenguaje Python [13]. Ofrece estructuras de datos y operaciones para la manipulación de tablas númericas y series temporales.

La elección de Pandas para la gestión de los datasets reside en la gran variedad de operaciones que ofrece para la manipulación y análisis de los mismos, que se combina a la perfección con la gran sencillez para la ejecución de las mismas.

3.1.4 Framework para construcción de API REST: Flask

Flask[7] [14] es un microframework web escrito en Python. En concreto se ha hecho uso de Flask-RESTful[8], una extensión para Flask que permite una rápida construcción de APIs REST alentando el uso de las mejores prácticas de desarrollo.

La elección de este framework se debe principalmente a que ofrece una forma sencilla de implementar un API REST, evitando realizar excesivas tareas de configuración y ofreciendo un resultado final adecuado a las necesidades del sistema. Además de las características anteriormente mencionadas junto su sencilla curva de aprendizaje han hecho que Flask se

[2] https://scikit-learn.org/
[3] https://www.tensorflow.org/
[4] https://pytorch.org/
[5] https://pandas.pydata.org/
[6] https://numpy.org/
[7] http://flask.pocoo.org
[8] http://flask-restful.readthedocs.io/en/latest

imponga a otras alternativas como Django[9], más potentes, pero con una complejidad mucho mayor.

3.2 API de Operaciones Básicas

El segundo módulo que trataremos se encarga de la gestión de los usuarios y de realizar consultas en base de datos que no requieren ningún análisis adicional. Este backend también expondrá un API REST que recibirá desde el frontal todas las peticiones relacionadas con la gestión de usuarios y las búsquedas de datasets y ejecuciones.

3.2.1 Lenguaje de programación: Java

Java[10] es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases. Este lenguaje de programación ha sido diseñado para permitir que el desarrollador escriba el programa una vez y pueda ejecutarlo en distintos dispositivos, sin necesidad de realizar una recompilación, es decir, una de sus máximas es la independencia de la plataforma.

La elección de este lenguaje para el desarrollo de este módulo del backend del sistema se fundamenta en el conocimiento sobre el mismo, su gran popularidad en la industria de desarrollo software en la actualidad, las aptitudes y *frameworks* que ofrece para la construcción de un módulo como el que se ha descrito y las facilidades que tanto el lenguaje como sus *frameworks* ofrecen para la programación. Concretamente se ha hecho uso de la Java Platform Enterprise Edition (Java EE) y de la versión Java 11 [15].

3.2.2 Framework web: Spring

Spring[11] [16] es un *framework* de código abierto para el desarrollo de aplicaciones web en Java. Este *framework* nos permite desarrollar aplicaciones de manera más rápida y eficaz evitando tareas repetitivas y ahorrando una gran cantidad de líneas de código. A continuación, mencionaremos los módulos ofrecidos por Spring que se han utilizado en el proyecto:

- **Spring Boot**[12] **[17]:** este módulo proporciona una infraestructura ligera que permite eliminar la mayor parte del trabajo de configuración de las aplicaciones basadas en Spring. Además permite compilar las aplicaciones como un archivo *.jar* que podremos ejecutar como una aplicación Java al uso.

[9] https://www.djangoproject.com/
[10] https://www.java.com
[11] https://spring.io/
[12] https://spring.io/projects/spring-boot

- **Spring Security**[13]**:** este módulo de Spring proporciona al desarrollador las herramientas necesarias para personalizar la autenticación y el control de acceso de sus aplicaciones.
- **Spring Data**[14]**:** este módulo ofrece una forma familiar y consistente para el acceso a distintos tipos de tecnologías de persistencia. En este proyecto se han usado los servicios proporcionados para JPA (Java Persistence API).

La elección de Spring como *framework* para el desarrollo de esta parte del sistema se debe principalmente a que, los módulos que hemos citado, junto con otros servicios que ofrece este *framework*, como la inyección de dependencias o la programación orientada a aspectos lo convierten en una solución perfecta para el desarrollo de una aplicación web Java como la que se requiere.

3.2.3 Framework para el desarrollo de pruebas: Mockito y JUnit

Para la construcción de un sistema seguro y fiable es fundamental que este cuente con un set de pruebas completo y que pruebe todos los aspectos del mismo. Por ello, y con el fin de facilitar la implementación de las pruebas se ha hecho uso de los *frameworks* Mockito[15] [18] o JUnit[16] [19].

Mockito es un *framework* que permite simular el comportamiento de distintos objetos con el fin de probar los distintos componentes del sistema de forma unitaria.

Por otro lado, Junit es el *framework* para realizar pruebas en Java más extendido, con este podremos ejecutar clases Java de forma controlada, evaluando su comportamiento.

3.3 Aplicación frontend

3.3.1 Lenguaje de programación: JavaScript

JavaScript[17] es un lenguaje de programación interpretado, creado a partir de ECMAScript. Es ligerom interpretado o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. JavaScript es un lenguaje de programación basado en prototipos, multi-paradigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional).

La elección de este lenguaje viene heredada de la elección del *framework* escogido para la elaboración del frontal de la aplicación web.

[13] https://spring.io/projects/spring-security
[14] https://spring.io/projects/spring-data
[15] https://site.mockito.org
[16] https://junit.org/junit5
[17] https://developer.mozilla.org/es/docs/Web/JavaScript

3.3.2 Framework: ReactJs

ReactJs[18] [20] es una biblioteca JavaScript *open source* diseñada para la elaboración de interfaces de usuario, más concretamente busca facilitar el desarrollo de las aplicaciones SPA. Es una librería declarativa y basada en componentes.

La elección de este *framework* radica en dos cuestiones principales. La primera ha sido el conocimiento previo del mismo, la segunda su capacidad para la construcción de PWAs (*Progressive Web Applications*). Una PWA es una SPA (*Single Page Application*) fiable, atractiva y que se adapta perfectamente a las capacidades del dispositivo en el que opera.

En contraposición con otros *frameworks* para el desarrollo de frontales como por ejemplo Thymeleaf[19] ofrece un mejor resultado en diferentes aspectos. Por un lado, ofrece un rendimiento mucho mejor ya que el renderizado se puede realizar por componentes sin necesidad de renderizar páginas completas. Además, permite reutilizar componentes de forma mucho más sencilla que los *frameworks* basados en plantillas. Por último, la experiencia de usuario que ofrecen las aplicaciones construidas con *frameworks* como React es mucho mejor que la ofrecida por los *frameworks* como Thymeleaf.

3.3.3 Biblioteca de diseño web: Material UI

Materail UI[20] [21] es una biblioteca que ofrece una serie de componentes de React con el fin de lograr un desarrollo web más rápido y sencillo. Los componentes que ofrece Material UI implementantan el sistema de diseño creado por Google, Material[21].

De esta forma, Material UI permite obtener unos resultados, en cuanto a diseño se refiere, muy buenos sin necesidad de elaborar de forma manual los estilos de todos los componentes del frontal de la aplicación.

3.4 Sistema de monitorización

Los módulos que hemos tratado harán uso de dos sistemas de monitorización. Uno de ellos se encargará del almacenamiento de los logs, así como de ofrecer una interfaz clara donde podremos consultar y realizar búsquedas sobre los mismos. El otro sistema registrará métricas sobre distintos aspectos del sistema, sobre las cuales podremos construir paneles para una sencilla y clara visualización de las mismas.

Ambos sistemas han sido desplegados en contenedores Docker, junto con las bases de datos necesarias para ambos y el motor de búsqueda utilizado por Graylog.

[18] https://es.reactjs.org/
[19] https://www.thymeleaf.org/
[20] https://material-ui.com/
[21] https://material.io

3.4.1 Docker

Docker[22] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Además, como se requiere desplegar varios contenedores se ha hecho uso de Docker Compose[23]. Compose es una herramienta para definir y ejecutar aplicaciones de varios contenedores Docker. Con Compose, se utiliza un archivo YAML para configurar los servicios de la aplicación. Luego, con un solo comando, creas e inicias todos los servicios de tu configuración.

3.4.2 Graylog

Graylog[24] es un sistema para el almacenamiento centralizado de logs que permite además realizar consultas sobre los datos, crear tablas con los resultados, crear alertas, etc.

3.4.3 Grafana

Grafana [25] es una herramienta que permite visualizar y formatear las métricas almacenadas de un sistema. Permite la creación de cuadros de mando y gráficos a partir de diversas fuentes.

3.5 Persistencia

En este apartado comentaremos las distintas soluciones de almacenamiento que se han empleado en el desarrollo del proyecto. Debido a la diversidad de datos almacenados se han utilizado soluciones bastante diversas para el almacenamiento de los mismos.

3.5.1 MySQL

Para el almacenamiento de los datos convencionales como los datos de los usuarios o algunos datos identificativos y descriptivos referentes a las ejecuciones y los datasets se ha decidido hacer uso de una base de datos relacional. Se ha valorado el uso de bases de datos como PostgreSQL[26] o MySQL[27].

Debido al conocimiento previo de MySQL por parte del alumno y a que ambas cumplen con los requisitos del sistema se ha optado por este sistema como gestor de base de datos.

[22] https://www.docker.com/
[23] https://docs.docker.com/compose/
[24] https://www.graylog.org/
[25] https://grafana.com/
[26] https://www.postgresql.org/
[27] https://www.mysql.com/

3.5.2 Almacenamiento de datasets y ejecuciones

Además, de los datos descriptivos e identificativos de los datasets y ejecuciones para ambos se ha requerido guardar ficheros, en el caso de los datasets se almacenan los propios datasets y en el caso de las ejecuciones se almacenarán los modelos resultantes del entrenamiento de los algoritmos. Para almacenar estos ficheros hemos habilitado un servidor que los almacena en un disco duro local organizado por tipo de fichero y usuario.

3.5.3 InfluxDB

InfluxDB[28] es el sistema de base de datos escogido para almacenar las métricas que consumirá el sistema de monitorización Grafana. InfluxDB es una TSDB (base de datos de series temporales) de código abierto desarrollada por InfluxData[29], diseñada para el almacenamiento rápido y con una alta disponibilidad de datos de series temporales, muy usado en campos como la monitorización y las métricas de las aplicaciones.

Una TSDB es una base de datos optimizada para manejar datos de series de tiempo, es decir, matrices de números indexados por tiempo.

3.5.4 MongoDB

Como hemos mencionado en el apartado anterior el sistema cuenta con un sistema de logs centralizado, para ello se almacenan los datos de los mismos en una base de datos MongoDB.

MongoDB[30] es un sistema de base de datos de código abierto NoSQL y basado en documentos. La combinación de esta tecnología con el motor de búsqueda Elasticsearch, que veremos a continuación, permite almacenar una gran cantidad de datos de log y consultarlos de forma rápida y eficiente.

3.5.5 Motor de búsqueda: Elasticsearch

Elasticsearch[31], como hemos mecionado, es el motor de búsqueda que usará Graylog para realizar consultas sobre los datos de log guardados en la base de datos NoSQL MongoDB. ElasticSearch está basado en Lucene[32] y provee un motor de búsqueda de texto completo, distribuido y con tenencia múltiple con una interfaz RESTful y con documentos JSON.

[28] https://www.influxdata.com/products/influxdb-overview/
[29] https://www.influxdata.com/
[30] https://www.mongodb.com/es
[31] https://www.elastic.co/es/
[32] https://lucene.apache.org/

3.6 Herramientas de soporte al desarrollo

Por último, cerraremos el capítulo dedicado a las tecnologías usadas en el proyecto dando unas pinceladas de las distintas herramientas que se han usado a lo largo del desarrollo del mismo para mejorar su calidad y facilitar el desarrollo.

3.6.1 Entornos de desarrollo

En lo referente a las herramientas que se usarán para el desarrollo de un proyecto, la elección de los entornos de desarrollo que se ajusten a las necesidades del proyecto es un factor fundamental para finalizarlo con éxito. La importancia de esta decisión reside en que la misma influye de forma directa en la productividad del desarrollador.

Un entorno intefrado de desarrollo o Integrated Development System (IDE) es una herramienta compuesta por un editor de código fuente sensible al lenguaje, un compilador y un depurador o *debugger*. Además, aportan facilidades para la definición, comprensión y gestión de los proyectos.

Por ello, con el fin de aumentar lo máximo posible la productividad en el desarrollo se ha hecho un estudio sobre los posibles entornos para desarrollar en cada uno de los lenguajes de programación con los que se construiría la plataforma. Para el desarrollo en lenguajes como Java y Python se valoró el uso de los IDE IntelliJ Idea[33] y PyCharm[34], ambos desarrollados por Jetbrains[35]. Ambas alternativas cumplen con los requisitos que se buscaban para el desarrollo del proyecto. Sin embargo sus versiones completas no son de código abierto, por lo que, finalmente, se ha optado por el uso de Eclipse como IDE para el desarrollo en Java y Python. A continuación, describiremos las características del mismo y justificaremos la elección realizada. Por otro lado, como ya hemos comentado el frontal de la plataforma ha sido desarrollado en JavaScript. Igual que en el caso anterior JetBrains ofrece una alternativa, cuya versión completa también es de pago, que se ajusta a las necesidades del proyecto, WebStorm[36]. Sin embargo, para el desarrollo en este lenguaje se ha optado por el uso de Visual Studio Code, un editor de código muy completo que como veremos en el apartado que le corresponde se ajusta a la perfección para las necesidades del proyecto.

[33] https://www.jetbrains.com/es-es/idea/
[34] https://www.jetbrains.com/es-es/pycharm/
[35] https://www.jetbrains.com/
[36] https://www.jetbrains.com/es-es/webstorm/

Eclipse

Eclipse[37] es un IDE *open source* y multilenguaje de los más empleados para el desarrollo en Java y Python. Cuenta con un *workspace* base y un sistema de *plugins* extendible que permiten personalizar el entorno de desarrollo según el gusto y las necesidades de cada desarrollador. Estas características unidas a la capacidad de integrar otras tecnologías usadas como el control de versiones y la inspección continua, convierten a este IDE en una solución ideal para el desarrollo del *backend* del proyecto. En lo relativo al desarrollo en Python se ha hecho uso de la perspectiva que proporciona el IDE PyDev[38].

Visual Studio Code

Visual Studio Code[39] no es considerado realmente un IDE ya que no cumple con las características esenciales que debe tener un entorno integrado de desarrollo. Sin embargo, es uno de los editores de código fuente más utilizados a día de hoy para el desarrollo debido a sus muchas virtudes. Lo bueno de VS Code es que consigue combinar perfectamente la simplicidad de un editor de código fuente con una serie de herramientas de desarrollo totalmente imprescindibles, como el autocompletado de IntelliSense, el control de versiones integrado o la depuración. Debido a todos estos factores es una de la herramientas más utilizadas para el desarrollo con el lenguaje de programación JavaScript. Por todos estos factores, VS Code ha sido el editor elegido para el desarrollo en JavaScript en el proyecto.

3.6.2 Control de versiones: Git

Un sistema de control de versiones es aquel que permite registrar los cambios realizados sobre un fichero o un conjunto de ficheros a lo a largo del tiempo, de tal manera que podemos recuperar las distintas versiones de los mismos. Por esto, es fundamental contar con uno para el desarrollo de un proyecto software.

Existen diversas alternativas para llevar a cabo esta tarea. Entre las más destacadas se encuentran Git, Subversion y Mercurial. Para el desarrollo del presente proyecto se ha escogido Git como sistema para el control de versiones.

Git[40] es un software de control de versiones distribuido y diseñado pensando en la eficiencia y la fiabilidad para el mantenimiento de versiones de aplicaciones que cuentan con un gran número de archivos de código fuente. La elección de este sistema por delante de los anteriormente mencionados se basa en la buena gestión de ramas que ofrece, el soporte por parte de la comunidad, el carácter distribuido que hemos comentado y el hecho de que se

[37] https://www.eclipse.org/
[38] http://www.pydev.org
[39] https://code.visualstudio.com/
[40] https://git-scm.com/

trata de un software libre. A pesar de que Mercurial, también cumple con las características descritas su elección ha sido descartada debido a que se trata de una tecnología no conocida para el alumno.

Además, se ha empleado la estrategia de Git-flow para la organización del repositorio. Esta estrategia se basa en la existencia de 5 tipos de ramas principales: *master, develop, release, feature* y *hot-fix*; además, cada funcionalidad se desarrollará en su rama correspondiente (normalmente el nombre de la misma va precedido de *feature/*), y una vez el desarrollo de la misma finaliza se realiza un *merge* a la rama *develop*. Por otro lado, cuando se proceda a realizar una entrega, por ejemplo al final de un Sprint, se crea una rama *release* a partir de la rama *develop*, sobre la cual se realizarán los tests pertinentes. De detectar errores en los mismos, los cambios se realizarán sobre la rama *release* correspondiente. Finalmente, tras finalizar las pruebas y resolver los errores detectados se realiza un *merge* de la rama *release* en *master* y *develop*. Por últmo, se usarán las ramas de *hot-fix* en caso de que sea necesario resolver errores encontrados en las entregas.

3.6.3 Gestor de proyectos: Taiga

Es fundamental contar una herramienta para la gestión de proyectos para el desarrollo de un proyecto de ingeniería, que permita mantener un registro de las tareas y del progreso de las mismas. En el mercado existen diversas opciones que cumplen con estas características: Redmine, Jira o Taiga.

La elegida para el desarrollo del proyecto tratado en este documento ha sido Taiga[41]. La facilidad de uso de esta herramienta, junto con la integración con otras tecnologías y su perfecta adaptación a la metodología de desarrollo, hacen de Taiga una herramienta perfecta para la gestión del proyecto a desarrollar.

3.6.4 Integración continua: Jenkins

La integración continua consiste en realizar la compilación y ejecución de pruebas del proyecto de forma frecuente, idealmente de forma diaria, con la meta de detectar posibles fallos lo más pronto posible.

Existen diversas herramientas que cubren esta necesidad, entre ellas destacan: Bamboo, Travis o TeamCity. Sin embargo, Jenkins sigue superando a las anteriores en varios aspectos y es por ello la herramienta de integración continua que se usará en el desarrollo del Trabajo de Fin de Grado.

Jenkins[42] es un servidor escrito en Java que permite la integración continua en cualquier proyecto. Es un software libre, que ofrece una sencilla configuración a través de trabajos cu-

[41] https://taiga.io/
[42] https://www.jenkins.io/

ya ejecución se puede configurar por ejemplo de forma periódica o tras realizar un cambio en el repositorio remoto. Además, es extensible, cuenta con un amplio conjunto de *plugins* que permiten su integración con otras muchas herramientas y se puede distribuir de forma sencilla.

3.6.5 Inspección continua

La inspección continua es un proceso holístico y completo que tiene como meta integrar la gestión de la calidad del código dentro del ciclo de vida del desarrollo software.

La gestión de la calidad consiste en medir la calidad interna del producto, es decir, evaluar su robustez, facilidad para su mantenimiento, etc.

SonarQube

La herramienta encargada de realizar la gestión de la calidad del código Java y Python ha sido SonarQube.

SonarQube[43] permite analizar y comprobar cual es el estado del proyecto en lo que a calidad de código se refiere a través de una amigable interfaz. Además, también provee sencillos mecanismos para su integración con otras herramientas usadas en el proyecto como Eclipse y Jenkins.

ESLint

Por otro lado, debido a que no se quería dejar de lado la calidad del código desarrollado en JavaScript se ha usado un *linter* adicional para inspeccionar este código. La herramienta escogida en este caso ha sido ESlint[44]. Esta herramienta de *linting* para JavaScript y JSX permite revisar el código tratando de encontrar errores que podrían provocar problemas de compilación o bien futuros *bugs* en nuestro desarrollo.

3.6.6 Documentación

En un proyecto con cierta entidad es fundamental contar con una buena documentación que apoye al código en la medida de lo posible. Por ello, se ha hecho uso de algunas herramientas que ayudan al mantenimiento y exposición de la misma.

[43] https://www.sonarqube.org/
[44] https://eslint.org/

Swagger

Swagger.io[45] ha sido la herramienta que se ha empleado para generar documentación sobre los APIs de la plataforma. Provee una interfaz clara que permite incluso realizar consultas contra las mismas.

DBDocs

DBDocs[46] es un software en desarrollo que permite generar documentación sobre las bases de datos de un sistema o plataforma.

[45] https://swagger.io/
[46] https://dbdocs.io/

Capítulo 4

Proceso de ingeniería

A lo largo de este capítulo se describe y se justifica la metodología de desarrollo sobre la cual se apoya el proceso de ingeniería empleado. La metodología escogida ha sido Scrum, sujeta a algunas adaptaciones, las cuales nos acercan más a la versión pragmática que a la técnica de esta metodología. Además, en este apartado trataremos también el resto de aspectos que conforman el proceso de ingeniería: descomposición en historias de usuario, estimación, gestión y organización del proyecto, planificación y gestión de riesgos.

4.1 Elección de la metodología

La elección de Scrum como la metodología a seguir en el proyecto ha sido el resultado del análisis y evaluación de las metodologías de desarrollo más utilizadas en la actualidad. Tras valorar las características, los requisitos y el tipo de gestión del proyecto, Scrum ha sido la metodología que más se adecuaba a las necesidades del mismo. Los criterios que se han utilizado para la toma de esta decisión han sido los siguientes:

- Rápida detección de problemas y riesgos, así como una rápida resolución de los mismos.
- *Feedback* del usuario de forma frecuente. Para la construcción de un a aplicación web consideramos muy importante que se ajuste a los criterios del usuario, por ello es necesario realizar entregas parciales del producto para que el usuario lo pueda utilizar e indicar posibles mejoras en el mismo.
- Gestión rápida y buena ante cambios en los requisitos. A lo largo del desarrollo de una plataforma como la que se plantea en el presente documento pueden surgir diferentes imprevistos que desemboquen en la aparición de nuevas funcionalidades o cambios en las existentes, por lo que es muy importante que la metodología escogida se adapte a dichos cambios.

Como se puede observar por los criterios arriba mencionados, una metodología ágil es la más adecuada para llevar a cabo la gestión del mismo. Éstas metodologías ofrecen una gran flexibilidad, una rápida adaptación e incorporación de los cambios que puede surgir durante el desarrollo. Se basan en entregas incrementales trabajando en iteraciones de entre una y cuatro semanas.

Por las razones comentadas, y como ya hemos adelantado anteriormente en este apartado la metodología escogida para llevar a cabo el proyecto ha sido Scrum. A continuación se explicará en detalle dicha metodología, así como las adaptaciones llevadas a cabo para que esta se ajustara correctamente a las necesidades del mismo.

4.2 Scrum

Scrum[1] es una metodología de desarrollo ágil nacida en los años 80 de la mano de Ikujiro Nonaka e Hirotaka Takeuchi. Scrum es el fruto del análisis del desarrollo de nuevos productos por parte de las principales empresas tecnológicas de la época. El nombre de Scrum nace debido a la comparativa que hicieron Nonaka y Takeuchi entre esta novedosa forma de trabajo y el avance de los jugadores de rugby en formación *scrum*. Así, una década más tarde, más concretamente en el año 1995, Ken Schwaber y Jeff Sutherland presentan *Scrum Development Process* [22] en la OOPSLA 95, un marco de reglas para el desarrollo software basadas en los principios de Scrum. Scrum se define, por lo tanto, como un marco de trabajo para productos complejos los cuales no pueden ser definidos de forma completa. Es decir, partimos de una orientación general para evolucionar hacia un definición más concreta. Por ello, el equipo deberá auto organizarse con el fin de determinar la mejor forma de entregar las funcionalidades que aporte mayor valor siguiendo las prioridades establecidas por el negocio.

4.2.1 Características

A continuación se detallarán las principales características de la metodología de desarrollo Scrum.

Iterativa e incremental

Scrum se basa en un conjunto de iteraciones, más comúnmente llamadas *sprints*. Un *sprint* es un *timebox* de entre 2 y 4 semanas en el cual se crea un incremento terminado del producto, utilizable y potencialmente desplegable. Este método de entregas parciales del producto permite evaluar el curso por el cual avanza el mismo en períodos de tiempo breves, ofreciendo así la posibilidad de realizar modificaciones tempranas de forma muy eficaz.

[1] https://www.scrum.org/

Gestión de riesgos y control de calidad

Un correcto uso de Scrum implica llevar a cabo una inspección frecuente y continua de los distintos artefactos así como del progreso hacia el objetivo establecido para la iteración. Asimismo, será necesario llevar a cabo una buena gestión de riesgos, es decir, se deben revisar los elementos que puedan implicar grandes desvíos en la planificación del proyecto. Estos seguimientos permitirán detectar posibles defectos y desviaciones de forma temprana, antes de que puedan convertirse en impedimentos relevantes.

Gestión de cambios y colaboración con el cliente

En Scrum es fundamental que todos los participantes en el proceso de desarrollo comprendan perfectamente todos los elementos del mismo. Esto permitirá que el cliente, el cual forma parte de este proceso, conozca de forma detallada la evolución del proyecto, así como los momentos en los que este será revisado. Por esto, el cliente tendrá la capacidad de proponer los cambios adecuados en el momento adecuado. Como consecuencia de lo anterior, en una metodología como Scrum es fundamental la capacidad de asimilar y adaptarse a los cambios propuestos. De esta forma, como habíamos indicado en la presentación de esta metodología, Scrum admite la incapacidad de definir y comprender el problema de forma completa en el momento de su concepción, teniendo que nutrirse de los nuevos cambios y requisitos propuestos a lo largo del desarrollo para completar su definición, llevando a cabo así una definición progresiva del producto.

Estas características convierten a Scrum en la metodología de desarrollo ágil más utilizada en la actualidad.

4.2.2 Estructura y adaptación de la metodología

Scrum define una serie de eventos, roles y artefactos que tendrán un impacto esencial en el resultado de la aplicación de la metodología. En la *Figura 4.1* se refleja, de forma esquemática, el flujo que sigue el desarrollo de un proyecto usando esta metodología. En primer lugar, se reflejan las funcionalidades o historias de usuario a realizar en el Product Backlog o Agenda el Producto, dichas historias de usuario se irán extrayendo del mismo para llevarlas a cabo en los *sprints*, que como ya hemos comentado tendrán una duración de entre 2 y 4 semanas, es importante que su duración sea constante sobre todo para poder llevar a cabo una buena estimación para los mismos. A lo largo de un Sprint, el producto pasará por las fases de análisis, diseño, implementación y pruebas. Por último, es importante destacar que el producto obtenido tras la finalización del sprint debe ser funcional y debe aportar valor al cliente.

Al comienzo de cada Sprint se realiza el Sprint Planning, en esta reunión el equipo Scrum al completo (Scrum Master y Product Owner incluidos) analizan, evalúan y repriorizan el

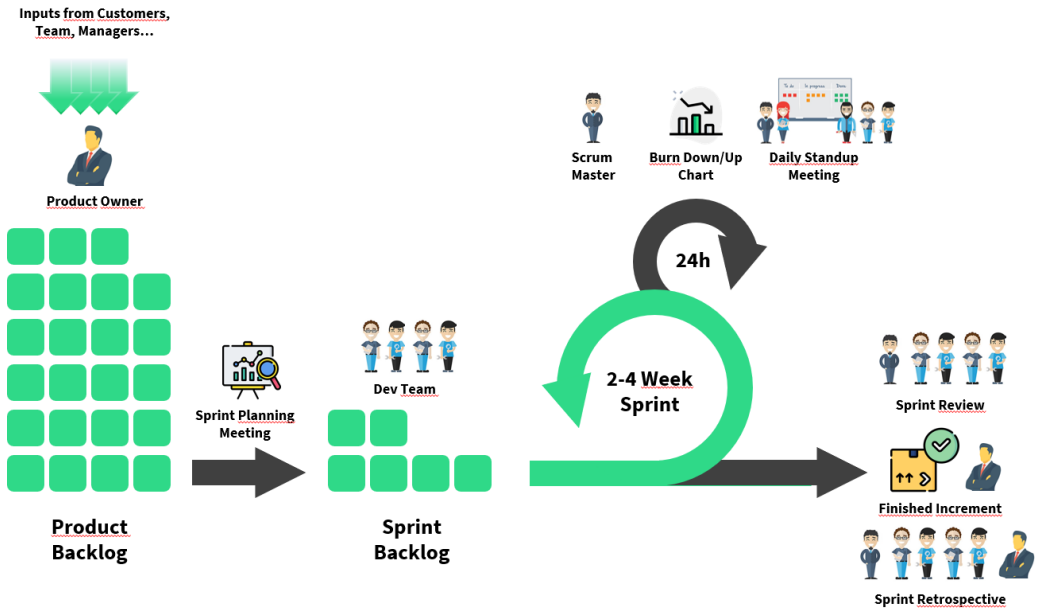

Figura 4.1: Esquema Scrum

Product Backlog para definir el objetivo del nuevo Sprint. Para ello se crearán las tareas correspondientes a las historias de usuario registradas en el Product Backlog, las cuales pasarán a formar parte del Sprint Backlog.

A lo largo del transcurso del Sprint el equipo de desarrollo llevará a cabo las tareas especificadas en el Sprint Backlog. Para llevar a cabo un buen seguimiento del Sprint, los miembros del equipo informarán diariamente y de forma individual sobre su progreso, trabajo futuro y dificultades encontradas al resto del equipo en las llamadas Daily Standup Meetings.

Finalmente, tras el fin de cada iteración o Sprint se llevan a cabo dos reuniones más, la Sprint Review y la Sprint Retrospective. En la primera el equipo analiza el trabajo realizado a lo largo del Sprint y determina en qué sería más óptimo seguir trabajando. Por otra lado, en la Sprint Retrospective el equipo analiza el modo en el que se ha realizado el trabajo a lo largo del Sprint, proponiendo mejoras para mejorar la forma en la que se realiza el trabajo de un Sprint.

Tras haber realizado una breve explicación del flujo habitual de Scrum, a continuación detallaremos cada uno de los componentes que conforman la metodología, así como la aplicación de cada uno de ellos en el proyecto que estamos tratando.

Eventos de Scrum

En Scrum existen un conjunto de eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de realizar reuniones no definidas por la metodología. Estos eventos pueden ser divididos en dos categorías, los Sprints y las reuniones, las cuales, como veremos a continuación, están estrechamente ligadas.

- **Sprint**

  El Sprint es el corazón de Scrum. Es un *timebox* cuya duración oscila entre las dos y las cuatro semanas que se repite de forma continua hasta la finalización del proyecto. La duración de cada Sprint se establece antes del comienzo del mismo y no puede cambiar una vez empezado, además es extremadamente recomendable que su duración sea constante a lo largo de todo el proyecto, esto permitirá obtener un mejor ritmo de trabajo y que las estimaciones sean más precisas.

  Como hemos dicho, no se debe cambiar la duración de un Sprint una vez comenzado, pero tampoco se deben introducir cambios en el objetivo del mismo una vez este ha comenzado.

  En el transcurso del Sprint el producto pasará por todas las fases del desarrollo y, al final del mismo, el equipo de desarrollo deberá presentar los avances realizados, ya que el incremento debería ser un producto potencialmente entregable al cliente.

  Tras analizar su viabilidad en lo referente al tamaño en puntos de historia de las historias de usuario identificadas al comienzo del proyecto, así como con el fin de obtener resultados tangibles con bastante frecuencia, se ha decidido que los Sprints contarán con una duración de dos semanas.

- **Reuniones**

  En Scrum se realizan cuatro tipos distintos de reuniones:

  - **Sprint Planning**

    Esta reunión tiene lugar al comienzo de cada Sprint, en ella, el equipo Scrum realizará un análisis del Product Backlog para determinar el objetivo a llevar a cabo a lo largo del Sprint. Tras determinar el objetivo del Sprint seleccionando las historias de usuario correspondientes se realiza la división en tareas de las mismas y se procede a su estimación. Por último, se toman las decisiones necesarias para lograr alcanzar el objetivo establecido.

  - **Daily Meeting**

    Como indica su propio nombre esta reunión se realiza diariamente, su duración debe ser breve (unos quince minutos aproximadamente). En ella cada miembro del

equipo informará al resto del equipo sobre su avance en sus tareas respondiendo a las siguientes preguntas:

  * ¿Qué hiciste ayer?
  * ¿Qué vas a hacer hoy?
  * ¿Qué impedimentos pueden surgir para el desarrollo de tus tareas?

Esta reunión permite por lo tanto que todo el equipo conozca cual es el estado del incremento en todo momento.

En el caso particular del proyecto que estamos tratando, el alumno será el único que tenga que responder a dichas preguntas, posibilitando así que la figura que representan los tutores conozcan de primera mano el avance del proyecto en todo momento.

- **Sprint Review**

  La Sprint Review se realiza al final de cada Sprint, en ella, se inspeccionará el incremento para conocer que partes del trabajo asignado al Sprint se han completado y cuales no. Además, se realizará una demo del incremento para demostrar el trabajo que se ha realizado.

- **Sprint Retrospective**

  De igual manera que la reunión anterior, esta también se lleva a cabo al final de cada Sprint, en esta, el equipo se inspecciona a sí mismo, expresando sus opiniones acerca de como ha transcurrido el Sprint con el fin de mejorar el proceso de trabajo. Para ello, el equipo responderá a las siguientes preguntas:

  * ¿Qué podemos comenzar a hacer?
  * ¿Qué debemos dejar de hacer?
  * ¿Qué debemos seguir haciendo?

En el proyecto que se trata en este documento, se han llevado a cabo las reuniones anteriormente mencionadas de la forma descrita, con el fin de tratar de aplicar de la mejor forma posible la metodología escogida.

Roles Scrum

Los roles mencionados a continuación representan la totalidad de los interventores en el proyecto.

Los equipos Scrum tienen dos cualidades fundamentales, la auto-organización y la multi-funcionalidad. Un equipo auto-organizado es aquel que no está dirigido por alguien externo al mismo, si no que se dirige a sí mismo y es el equipo quien debe elegir la mejor forma de llevar

a cabo el trabajo. Por otro lado, la multifuncinalidad de un equipo Scrum radica en la capacidad del equipo para llevar a cabo cualquier tarea necesaria para el proyecto sin necesidad de depender de agentes externos al mismo.

Los roles que define la metodología son: el Product Owner, el Scrum Master y el Development Team. Esta estructura busca optimizar la flexibilidad, la creatividad y la productividad.

- **Product Owner**

  El Product Owner es el encargado de maximizar el valor de producto y el trabajo del Development Team. Por ello, es la figura responsable de gestionar el Product Backlog, asuminedo la responsabilidad de definir y priorizar las funcionalidades o historias de usuario, además tras cada iteración deberá ajustar las estimaciones y repriorizar el Product Backlog según las necesidades del negocio. Por último, el Product Owner tiene la potestad para decidir las fechas en las que se debe entregar cada incremento, así como para valorar y aceptar o rechazar el resultado obtenido tras un Sprint.Debido a lo comentado anteriormente, el Product Owner será quien represente al cliente dentro del equipo Scrum.

  Este rol será asumido por los directores de Trabajo de Fin de Grado.

- **Scrum Master**

  El Scrum Master es el encargado de que la metodología se entienda y se aplique correctamente en el proyecto. Por ello debe promover las buenas prácticas de la misma, pero siempre sin descuidar su papel como desarrollador. Además, el Scrum Master es la persona encargada de representar al equipo frente a la dirección y viceversa, protegiéndolo de los impedimentos externos y actuando como escudo frente a las distintas interferencias externas que pudieran surgir. El Scrum Master, por lo tanto, proporciona un servicio a los desarrolladores.

  Debido a la gran complejidad para encontrar una figura que se adapte correctamente a este rol para el proyecto, hará que se adopte una versión de la metodología más pragmática que se acerca más al concepto original de Scrum, imponiendo los valores y resultados por encima de las reglas definidas. Por esto, serán los directores del Trabajo de Fin de Grado los encargados de abogar por el cumplimientos de la metodología y de eliminar los impedimentos externos que pudieran surgir, sin la necesidad de ocupar el rol de Scrum Master al uso.

- **Development Team**

  El equipo de desarrollo debe estar formado, idealmente, por un grupo de entre 4 y 9 personas. Estos equipos deben cumplir dos requisitos fundamentales para poder llevar

a cabo el proceso establecido en la metodología, deben ser auto-organizativos, es decir, deben tener la capacidad y autoridad para decidir como realizar el trabajo; y de deben ser multifuncionales, es decir, deben contar con todos los perfiles y capacidades necesarios para realizar cualquier trabajo para el desarrollo del producto, sin necesidad de ayuda externa.

Idealmente, la dedicación del Develepment Team debe ser a tiempo completo, además, no debe haber cambios en la composición del mismo en el transcurso de un Sprint.

Los integrantes de este eqipo tienen la responsabilidad de desglosar las historias de usuario en tareas y realizar la estimación de las mismas, además, deben reportar al resto del equipo su progreso y problemas detectados en el desarrollo de las mismas en las reuniones habilitadas para ello. Por último, deben realizar el desarrollo de las tareas asignadas a cada Sprint para conseguir un incremento funcional y que aporte valor al final del mismo.

La figura del Development Team, debido al carácter del proyecto estará representada por una única persona, el alumno, quien realizará el desarrollo del Trabajo de Fin de Grado. Una vez más podemos apreciar la necesidad de aplicar una versión de Scrum más pragmática.

Artefactos de Scrum

- **Product Backlog**

  El Product Backlog es una lista priorizada de todo lo que es necesario para desarrollar el producto, así como la única fuente de requisitos para cualquier cambio a realizar sobre el mismo. Todas las entradas de esta lista deben aportar valor al cliente.

  Como ya hemos comentado en su apartado, el encargado de su gestión es el Product Owner, por ello, debe gestionar su contenido, disponibilidad y ordenación.

  El Product Backlog es un artefacto dinámico, es decir, cambia constantemente para adaptarse a lo que el producto necesita en cada instante. Además, cabe destacar que el Product Backlog nunca se considera completo, si no que evoluciona a medida que evolucionan el producto y su entorno.

  Para este proyecto en concreto, el Trasbajo de Fin de Grado, el Product Backlog ha sido definido en su mayoría al comienzo del proyecto ya que es necesario conocer los objetivos, herramientas y el plan de trabajo del mismo para su especificación previa en el anteproyecto.

- **Sprint Backlog**

  El Sprint Backlog es el conjunto de entradas del Product Backlog seleccionadas para un determinado Sprint. Estas estarán divididas en tareas estimadas acompañadas por un plan de trabajo para la entrega del incremento y la consecución del objetivo.

  Este artefacto hace visible el tiempo que el Equipo de Desarrollo percibe como necesario para alcanzar el objetivo establecido.

  Como ya hemos mencionado, las tareas que lo componen están estimadas. Esta estimación será realizada en horas y esta nunca debería superar las dieciséis horas ya que deben ser lo suficientemente pequeñas para poder detectar inconvenientes en el progreso en las reuniones diarias.

- **Burndown Charts**

  Un *burndown chart* es una representación gráfica del trabajo pendiente de hacer enfrentado con el tiempo. Normalmente, el trabajo pendiente se representa en el eje vertical mientras que el tiempo se representa en el eje horizontal.

  Este artefacto es muy útil para predecir cuando se completará el trabajo, pudiendo ver como afectarían a las fechas de entrega los cambios en los requisitos o en el equipo de desarrollo.

La gestión de todos los artefactos que hemos comentado se llevará a cabo mediante la herramienta de gestión de proyectos Taiga.

4.3 Gestión del proyecto

4.3.1 Product Backlog

En esta sección se estudiarán las historias de usuario que conforman el Product Backlog.

- **Trabajo previo**

  - **H1: Análisis de modelos de clasificación y herramientas para su implementación**

    Lectura y análisis de documentación sobre los distintos modelos de clasificación y sobre las distintas tecnologías y herramientas usadas para su implementación.

  - **H2: Análisis de viabilidad de las tecnologías y metodologías**

    Estudio de las ventajas y desventajas de distintas metodologías aplicables al desarrollo software, así como de las posibles tecnologías que encajan en el proyecto a desarrollar,

  - **H3: Instalación y configuración de las herramientas para el desarrollo**

    Instalación y configuración en la máquina del alumno de las distintas herramientas necesarias para completar el dearrollo del proyecto.

- **API de Machine Learning**

  - **H4: Almacenamiento de datasets**

    Almacenamiento de los *datasets* subidos a la plataforma por parte de los usuarios en el sistema de ficheros donde corre la aplicación.

  - **H5: Modificación de un dataset**

    Actualización de los datos de los *datasets*, incluido los ficheros que lo conforman.

  - **H6: Análisis detallado de un dataset**

    Análisis de las *features* de los *datasets* subidos por los usuarios siempre y cuando estos cumplan los estándares de validez establecidos en la plataforma.

  - **H7: Ejecución de modelos de clasificación**

    Ejecución de modelos de clasificación sobre aquello *datasets* considerados válidos por la plataforma. Se ofrecerán una serie de algoritmos predefinidos, y se otorgará al usuario la capacidad de parametrizarlos, así como de elegir el porcentaje de datos del *dataset* usados para test y entrenamiento. Finalmente, tras hacer un estudio de los *datasets* ofrecidos por distintas fuentes se ha decidido que los *datasets* deben ser un único fichero CSV para poder ser ejecutados o analizados. Esta decisión se toma tras comprobar que la mayoría de proveedores de *datasets* siguen este formato que será el más amigable para los usuarios. Además, como se plantea en el último apartado (6.3 Trabajo futuro), se deja abierta la posibilidad de ampliar el espectro para ejecutar sobre *datasets* que sigan otros formatos.

    Además, se ofrecerá la capacidad de indicar un presupuesto en términos de tiempo de CPU para la ejecución de la clasificación.

  - **H8: Extracción de *features* durante la clasificación**

    Además de lo especificado anteriormente, el usuario podrá especificar un método de extracción de características entre los ofrecidos por la plataforma y de forma opcional para que este se ejecute de forma previa a la clasificación.

  - **H9: Almacenar modelo de clasificación resultante**

    Por último, dentro de las historias correspondientes a la API de Machine Learning, se almacenará, de nuevo en el sistema de ficheros, el modelo de clasificación resultante de una ejecución exitosa, con el fin de que el usuario lo pueda descargar y ejecutar de forma local sobre otros *datasets*.

- **API de Operaciones Básicas**
  - **H10: Búsqueda de *datasets* por keywords**

    El usuario podrá realizar búsquedas de *datasets* por palabras clave. Las búsquedas deben devolver los resultados paginados, asegurando así un mejor rendimiento en el sistema.
  - **H11: Búsqueda de *datasets* de un usuario**

    El sistema permitirá buscar los *datasets* de un usuario. La búsqueda devolverá los resultados paginados para un mejor rendimiento.
  - **H12: Búsqueda de ejecuciones de un usuario**

    El sistema permitirá buscar las ejecuciones de un usuario. La búsqueda devolverá los resultados paginados para un mejor rendimiento.
  - **H13: Búsqueda de ejecuciones por keywords**

    El usuario podrá realizar búsquedas de ejecuciones por palabras clave. Las búsquedas deben devolver los resultados paginados, asegurando así un mejor rendimiento en el sistema.
  - **H14: Búsqueda de ejecuciones similares a otra**

    El sistema mostrará las ejecuciones similares a otra. La similitud se basará en el *dataset* y el algoritmo usado para la misma.
  - **H15: Descargar *dataset***

    Los usuarios podrán descargar los *datasets* subidos a la plataforma.
  - **H16: Descargar modelo de clasificación**

    Los usuarios podrán descargar el modelo de clasificación resultante de una ejecución determinada.
  - **H17: Descargar predicciones de una clasificación**

    Los usuarios podrán descargar el *dataset* con las predicciones sobre los datos de test.
  - **H18: Registrar usuario**

    La plataforma permitirá dar de alta usuarios en el sistema.
  - **H19: Iniciar sesión**

    Un usuario podrá iniciar sesión usando su nombre de usuario y contraseña.
  - **H20: Cerrar sesión**

    Un usuario autenticado podrá cerrar sesión.
  - **H21: Modificar información personal de un usuario**

    Un usuario podrá modificar su información personal.

  - **H22: Cambiar contraseña**

    Los usuarios pueden cambiar su contraseña. Deberán indicar la contraseña antigua y la nueva contraseña a utilizar.

  - **H23: Mostrar información de un usuario**

    Los usuarios podrán ver su información personal almacenada en el sistema.

- **Configuración y puesta a punto de herramientas para la monitorización del sistema**

  - **H24: Set-up de Docker en una máquina externa en la que se lanzarán los servicios de monitorización**

    Se levantará un Docker en una máquina externa en la misma red donde se montarán los servicios de monitorización que emplea la plataforma.

  - **H25: Set-up de la base de datos y el motor de búsqueda en Docker para el sistema de logs**

    Se dará de alta una base de datos documental y un motor de búsqueda en el Docker para el almacenamiento y búsqueda de logs.

  - **H26: Set-up de la base de datos en Docker para el sistema de métricas**

    Se dará de alta una base de datos en Docker para almacenar los datos de métricas de la plataforma.

  - **H27: Set-up de Graylog en Docker**

    Se levantará en Graylog en Docker para que haga uso de la base de datos y motor de búsqueda indicados. Además, se configurarán los proyectos para que guarden los logs de forma adecuada para que se muestren en Graylog.

  - **H28: Set-up de Grafana en Docker**

    Se levantará Grafana en Docker para que haga uso de la base de datos indicada. Además, se configurará el proyecto Java para que reporte datos de métricas que posteriormente se mostrarán en Grafana.

4.3.2 Recursos

El proyecto necesitará los siguientes roles para su desarrollo:

- Product Owner (PO)
- Jefe de proyecto (JP)
- Investigador (I)

- Analista (A)
- Diseñador (D)
- Programador (P)

Al tratarse este proyecto de un Trabajo de Fin de Grado, todos los roles serán desarrollados por parte del alumno, a excepción del rol de PO, que será asumido por los directores del mismo.

4.3.3 Asignación de recursos a historias

Tras la definición de los recursos y las historias de usuario se procede a la asignación de los recursos a estas. Esta asignación se realizará según lo indicado en la *Tabla 4.1*.

Historia	Recursos	Historia	Recursos	Historia	Recursos
H1	I, A, P	H11	P	H21	P
H2	PO, JP, A	H12	P	H22	P
H3	P	H13	P	H23	P
H4	A, D, P	H14	A, P	H24	I, P
H5	P	H15	A, D, P	H25	I, P
H6	A, D, P	H16	A, D, P	H26	I, P
H7	A, D, P	H17	A, D, P	H27	I, P
H8	A, D, P	H18	P	H28	I, P
H9	P	H19	P		
H10	P	H20	P		

Tabla 4.1: Asignación de recursos humanos a historias de usuario.

4.3.4 Estimación de tiempos

Habitualmente, la estimación de las historias en la metodología Scrum se realiza mediante puntos de historia, los cuales, representan la cantidad de trabajo que supone una historia o tarea al equipo de trabajo. Sin embargo, para llevar a cabo un buen uso de los mismos se requiere un equipo que haya trabajado junto de forma previa usando este método de estimación. Por esta razón, y debido a las circunstancias del proyecto se ha decidido realizar una estimación según la cual cada punto de historia se corresponda con una hora de trabajo.

En la *Tabla 4.2* podemos ver la estimación en puntos de historia para cada historia de usuario.

4.3.5 Desarrollo del proyecto

Las historias de usuario descritas anteriormente se han agrupado en un total de 10 Sprints, de dos semanas cada uno. La disminución de la productividad en algunos períodos se debe

Historia	Horas	Historia	Horas	Historia	Horas
H1	40	H11	8	H21	8
H2	13	H12	8	H22	5
H3	13	H13	13	H23	8
H4	40	H14	20	H24	30
H5	8	H15	30	H25	10
H6	40	H16	20	H26	10
H7	40	H17	20	H27	10
H8	30	H18	13	H28	10
H9	8	H19	10		
H10	13	H20	5		
Total					483

Tabla 4.2: Estimación de tiempo en horas para cada historia.

principalmente a motivos académicos y profesionales, como la realización de prácticas en empresa, entrega de prácticas del grado y finalmente el comienzo del alumno en el mundo laboral. Además, para algunas de las tareas el tiempo estimado no se ha correspondido finalmente con el necesario para el desarrollo de las mismas.

Todos estos datos pueden apreciarse en el *burndown chart* del proyecto sacado de la herramienta de gestión de proyectos utilizada: Taiga; y expuesto en la *Figura 4.2*.

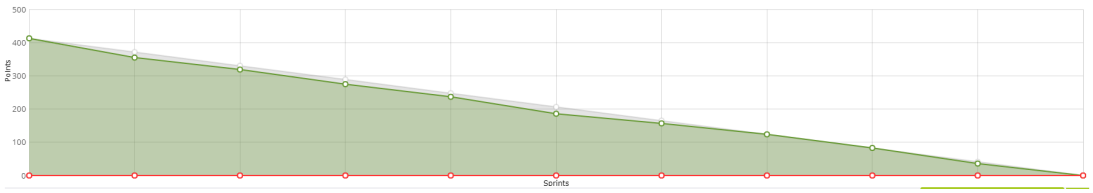

Figura 4.2: Burndown chart del proyecto

Capítulo 5

Desarrollo

A lo largo de este capítulo analizaremos con detalle todo el proceso que se ha seguido para el desarrollo del proyecto. Por ello, comenzaremos realizando el análisis de requisitos, a partir del mismo realizaremos el diseño de la arquitectura de la plataforma junto con el análisis, el diseño y la implementación de los componentes. Por último, comentaremos el juego de pruebas diseñado y desarrollado junto con las herramientas de ayuda al desarrollo utilizadas, así como las herramientas de monitorización usadas.

5.1 Análisis de Requisitos

Para conseguir finalizar un proyecto software de forma exitosa y conseguir un producto con las características deseadas es fundamental realizar una buena identificación y análisis de los requisitos.

Por ello, en esta sección especificaremos los requisitos funcionales y no funcionales que se han identificado para el presente proyecto.

5.1.1 Requisitos funcionales

Los requisitos funcionales definen las funciones que debe cumplir el sistema o sus componentes. Los requisitos funcionales identificados en este proyecto son los siguientes:

- R1 - Gestión de usuarios
- R2 - Gestión de *datasets*
- R3 - Análisis de *datasets*
- R4 - Ejecución de clasificaciones
- R5 - Descarga de modelos de clasificación
- R6 - Mostrar resultados de clasificación

5.1.2 Requisitos no funcionales

Los requisitos no funcionales no se refieren directamente a las funciones específicas suministradas por el sistema, sino a las propiedades del mismo: rendimiento, seguridad, disponibilidad. Estos afectarán de forma directa en las decisiones de diseño, implementación y en las decisiones tecnológicas del mismo. Los requisitos no funcionales identificados en este proyecto son los siguientes:

- Usabilidad
- Escalabilidad
- Mantenibilidad
- Alta disponibilidad

5.2 Arquitectura del sistema

Tras realizar la extracción y el análisis de los requisitos especificados en el apartado *5.1 Análisis de Requisitos* se realiza el diseño de la arquitectura de la plataforma. En este diseño se identifican 5 componentes principales: el API de Machine Learning, el API de operaciones básicas, el frontal del sistema, el sistema de monitorización y el sistema de almacenamiento.

API de Machine Learning

De forma global este componente se encargará de los aspectos relacionados con el Machine Learning, es decir, ejecución de clasificaciones y gestión y análisis de *datasets*. Por ello, este API permitirá subir y actualizar datasets, obtener una visualización previa del mismo y ejecutar clasificaciones sobre los mismos.

API de operaciones básicas

Por otro lado, el sistema cuenta con este segundo API el cual ofrecerá al usuario diferentes operaciones básicas sobre sus propios datos, así como sobre distintos datos de los datasets y ejecuciones. Por ello, meidante el uso del mismo se realizarán todas las operaciones relacionadas con la gestión de usuarios, búsquedas de datasets y ejecuciones, y descarga de datasets y modelos de clasificación.

Frontal web

Este componente se encargará de ofrecer al usuario una interfaz web *responsive* a través de la cual pueda interactuar con los APIs descritos anteriormente.

Sistema de monitorización

Además de los tres componentes principales comentados anteriormente y el componente de almacenamiento que comentaremos a continuación la plataforma cuenta con un sistema de monitorización. Este sistema se divide en dos subsistemas: sistema de *logs* (Graylog) y sistema de representación de métricas (Grafana).

Almacenamiento

Este componente se encarga de la gestión de la persistencia de todos los datos necesarios en la plataforma. Podemos dividir este componente en dos subcomponentes: el almacenamiento principal y el almacenamiento de datos de monitorización.

En primer lugar, en cuanto al almacenamiento principal la plataforma cuenta con una base de datos SQL (MySQL) en la cual se persistirán todos los datos relacionados con los usuarios y los datos descriptivos de los *datasets* y de las ejecuciones. Por otro lado, el almacenamiento de los *datasets* y de los ficheros donde se guardan los modelos resultantes de las ejecuciones se realizará directamente en disco, sin hacer uso de una base de datos.

Por otro lado, el almacenamiento de los datos de monitorización se puede dividir en dos subcomponentes. Los datos de *logs* se guardarán en una base de datos documental MongoDB, además este sistema usará el motor de búsqueda Elasticsearch para realizar búsquedas sobre esta base de datos. Por otro lado los datos de métricas serán almacenados en una base de datos de series temporales, InfluxDB.

5.2.1 Esquema general del sistema

En la *Figura 5.1* podemos ver el diagrama general de la arquitectura descrita anteriormente.

5.2.2 Modelo de datos SQL

En la *Figura 5.2* podemos ver el modelo de datos SQL (obtenido a través de la herramienta de documentación DBDocs) que consumirán ambos APIs.

5.3 Machine learning API

En esta sección trataremos todos los aspectos relacionados con el componente que se encarga de la ejecución de las clasificaciones y la gestión de los datasets.

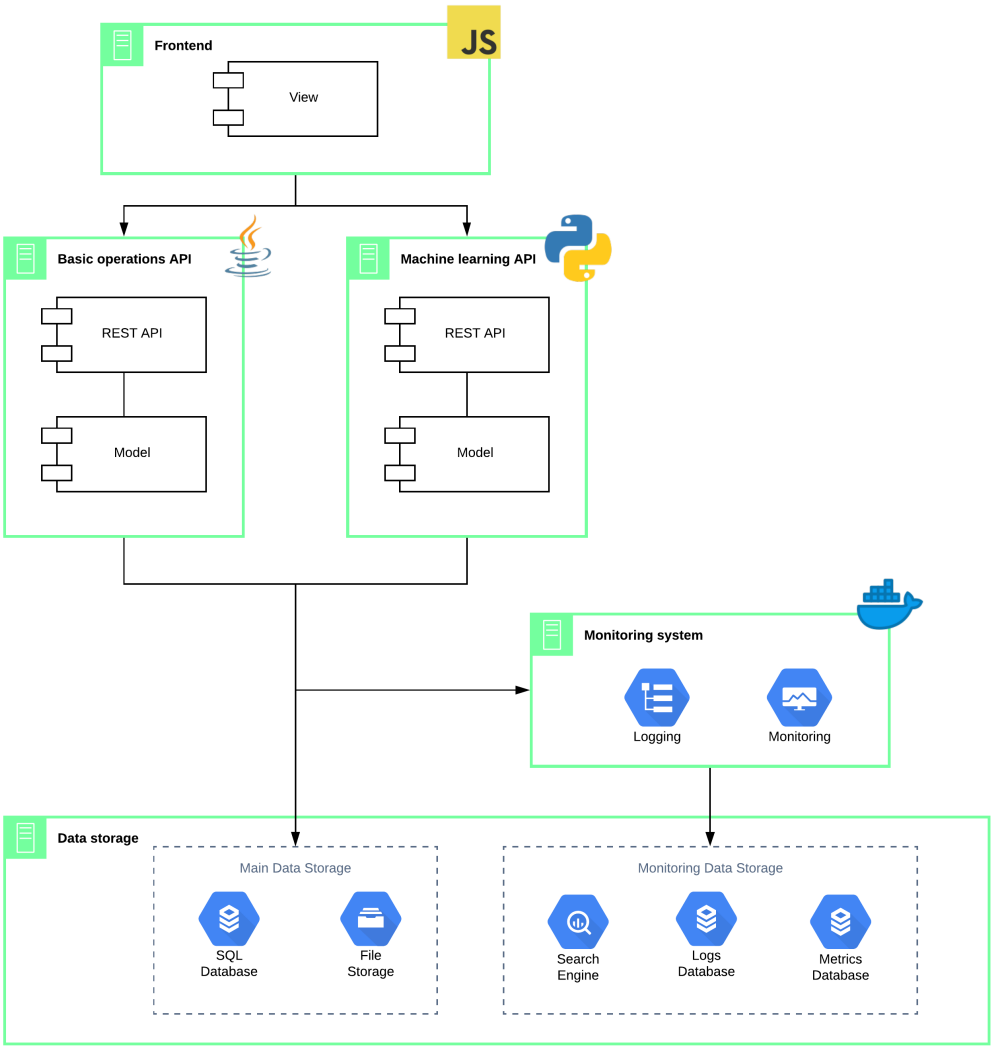

Figura 5.1: Arquitectura global

5.3.1 Análisis

Este componente se hará cargo de los requisitos «R3 Análisis de *datasets*» y «R4 Ejecución de clasificaciones». Además, se hará cargo de parte del requisito «R2 Gestión de datasets», siendo el encargado de almacenarlos en base de datos. A continuación se detalla el análisis de este componente en base a los actores que hacen uso del mismo y los casos de uso que lo conforman.

Actores del componente

En este apartado se detallan los actores que podrán hacer uso del componente:

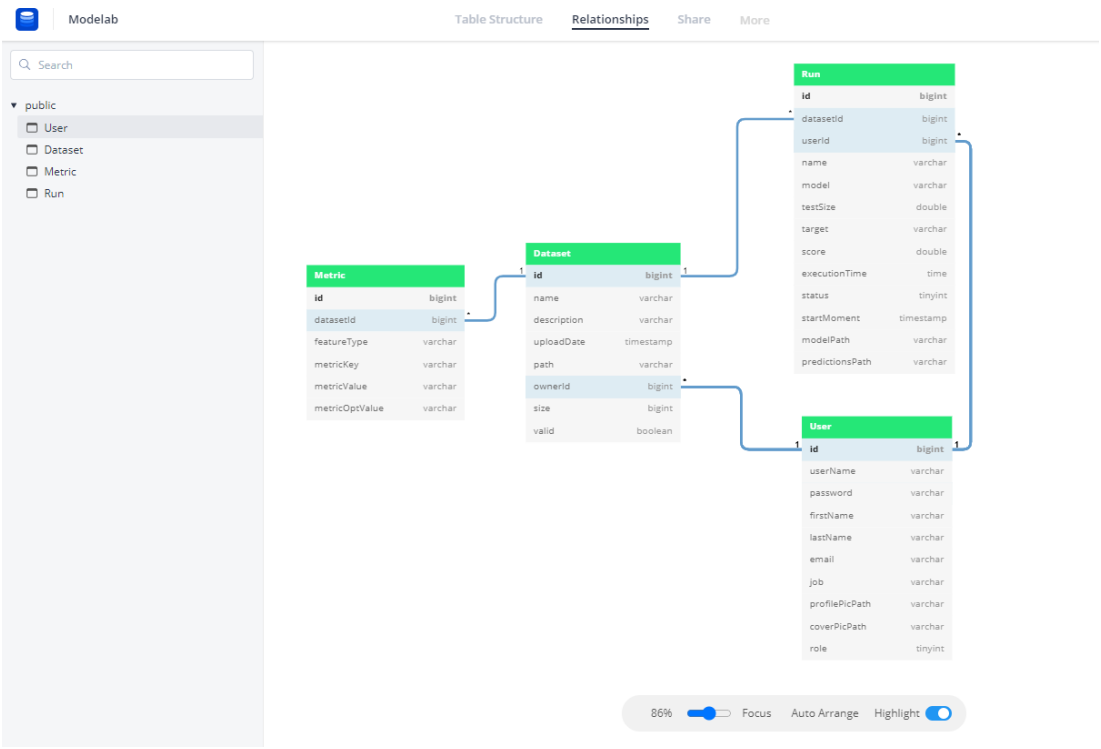

Figura 5.2: Modelo de datos SQL - Fuente https://dbdocs.io/JorgeGabin/Modelab

- **A1 Usuario no autenticado:** Este tipo de actor tendrá acceso de forma limitada a las operaciones ofrecidas por el API. Por ello, solo tendrá acceso a las operaciones de obtener la vista previa de un *dataset* y obtener las columnas del mismo.

- **A2 Usuario autenticado:** El actor usuario autenticado tendrá acceso a todas las operaciones que ofrece el API, es decir, podrá realizar las operaciones especificadas para un usuario no autenticado y además podrá: subir y modificar *datasets*, y lanzar ejecuciones.

A continuación podremos ver reflejados dichos permisos en el diagrama de casos de uso del componente.

Casos de uso

A partir de los requisitos que debe cubrir el componente se extraen los casos de uso pertinentes.

En la *Figura 5.3*, se muestra el diagrama de casos de uso correspondiente al componente tratado en este apartado.

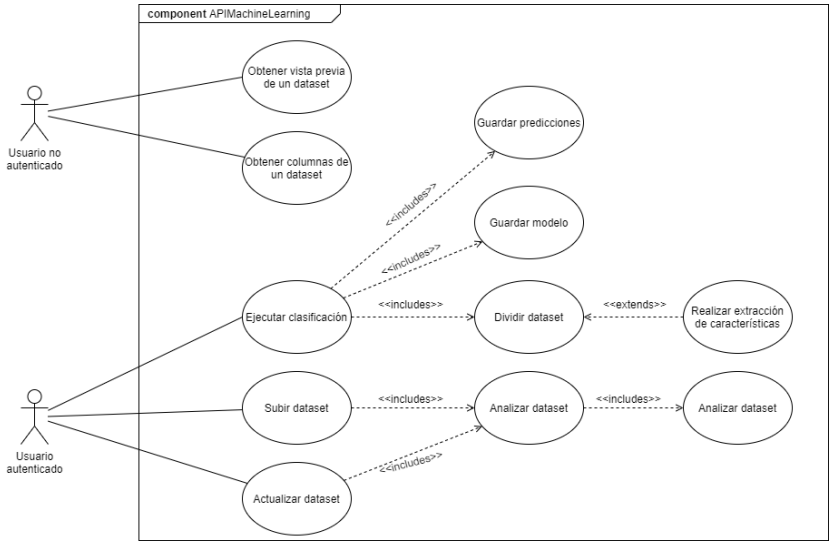

Figura 5.3: Diagrama de Casos de Uso API Machinie Learning

5.3.2 Diseño e implementación

En esta sección comentaremos el diseño que se ha seguido para el desarrollo del API de Machine Learning. Para ilustrar la estructura del componente haremos uso de diagramas UML.

Arquitectura

Para estructurar el API hemos decidido implementar una arquitectura por capas sencilla en la que se dividen las responsabilidades entre los controladores, que se encargarán de exponer los *endpoints* del API; y el modelo, que realizará las labores de acceso a datos y lógica de negocio.

Estructura del componente

La distribución en paquetes que se ha seguido para el API es la siguiente:

- **com.cologic.modelab.ml-api.model:** Este paquete incluye los servicios y clases encargados de la lógica de negocio y del acceso a datos.
- **com.cologic.modelab.ml-api.rest:** En este paquete se encuentran las clases encargadas de la implementación de los dos controladores que expondrán las diferentes operaciones del API al exterior.
- **com.cologic.modelab.ml-api.utils:** Contiene una serie de utilidades comunes a ambas capas.

Puntos clave

En este apartado comentaremos los puntos clave en el diseño y desarrollo del componente:

- **Ejecución de clasificaciones:** La funcionalidad principal del sistema recae en la ejecución de clasificaciones, responsabilidad de este componente. Para llevar a cabo esta labor, como habíamos comentado, se ha elegido el *framework* Scikit-learn que permite realizar esta tarea de forma sencilla y con un buen rendimiento.

  A continuación, se muestra un ejemplo de como se emplea el *framework* para realizar el entrenamiento de un modelo, y el cálculo de predicciones y puntuación del mismo.

```
def StochasticGradientDescent(self, x_train, x_test, y_train,
    y_test, params):

        [...]

        # Create Stochastic Gradient Descent classifier
        sgd = SGDClassifier(loss=params.get('loss'),
            shuffle=params.get('shuffle'),
            random_state=params.get('randomState'))
        # Fit the classifier to the data
        sgd.fit(x_train, y_train)

        # Save trained model
        model_path = utils.save_model(sgd)

        # Predict on all test data
        predictions = sgd.predict(x_test)
        predictions_path = utils.save_predictions(predictions,
            x_test, y_test)

        # Get model score
        score = sgd.score(x_test, y_test)

        [...]
```

Fragmento de código 5.1: Ejecución del algotirmo Stochastic Gradient Descent sobre un dataset.

- **Extracción de características:** Otro aspecto fundamental previo a la ejecución de una clasificación es la extracción de *features*, que como hemos explicado disminuye en gran medida los recursos necesarios para dichas ejecuciones. Para esta labor, de nuevo se hace uso del *framework* Scikit-learn que ofrece una serie de métodos de extracción de

características que se ejecutarán a petición del usuario sobre los datos para el entrenamiento del modelo.

- **Presupuesto en tiempo de ejecución para las clasificaciones:** Además, el sistema como hemos comentado permite al usuario establecer un presupuesto en términos de tiempo de CPU para las ejecuciones. Por ello, las clasificaciones contarán con un *timeout* opcional. Para la implementación de dicho *timeout* se ha hecho uso de la anotación *@stopit.threading_timeoutable* ofrecida por la librería *stopit*[1] que permite pasar un parámetro a la función indicando el timeout de la misma.

- **Extracción de métricas de los *datasets*:** Con el fin de ofrecer a los usuarios la máxima cantidad de información posible acerca de los *datasets*, durante su subida a la plataforma, y si estos cumplen los estándares de validez de la misma, se realizará un análisis de las características de los mismos según su tipo de dato. Para sacar las métricas de cada una de ellas, y después mostrarlas de forma gráfica al usuario, se ha hecho uso de las distintas posibilidades que ofrece la librería Pandas.

- **Controladores:** Para la implementación de los controladores, como ya se ha comentado en el Capítulo 3, se ha usado el *framework* Flask. Este *framework* permite la construcción de un API REST de forma rápida y sencilla sin necesidad de realizar demasiadas configuraciones.

  En el fragmento de código 5.2 podemos observar como se realizaría la exposición de uno de los *endpoints* del API.

```
1 @datasets_api.route('/api/ml/dataset/preview/<dataset_id>',
      methods=['GET', 'OPTIONS'])
2 @cross_origin()
3 def get_dataset_preview(dataset_id):
4
5     response = datasetService.get_dataset_preview(dataset_id)
6
7     return jsonify(response)
```

Fragmento de código 5.2: Ejemplo de un endpoint del controlador de datasets.

5.4 API Operaciones básicas

En esta sección se tratan todos los aspectos relacionados con el desarrollo de la API de operaciones básicas, la cual se encarga de la gestión de los usuarios, las búsquedas de *datasets* y ejecuciones, y por último de la descarga de *datasets* y modelos de clasificación.

[1] https://pypi.org/project/stopit/

5.4.1 Análisis

Este componente se encarga de los requisitos «R1 Gestión de usuarios», «R5 Descarga de modelos de clasificación», «R6 Mostrar resultados de clasificación» (los resultados serán almacenados en base de datos durante la ejecución de las clasificaciones) y los aspectos restantes del requisito «R3 Gestión de *datasets*». A continuación se detalla el análisis de este componente en base a los actores que hacen uso del mismo y los casos de uso que lo conforman.

Actores del componente

Los actores que harán uso del componente serán los mismos que en el comentado anteriormente.

A continuación podremos ver reflejados dichos permisos en el diagrama de casos de uso del componente.

Casos de uso

A partir de los requisitos que debe cubrir el componente se extraen los casos de uso pertinentes.

En las *Figuras 5.4 y 5.5*, se muestran los diagramas de casos de uso correspondientes al componente tratado en este apartado.

5.4.2 Diseño e implementación

En esta sección comentaremos el diseño que se ha seguido para el desarrollo del API de Operaciones Básicas. Para ilustrar la estructura del componente haremos uso de diagramas UML.

Arquitectura

Para estructurar el API seguiremos una arquitectura por capas. Concretamente seguiremos el patrón Model-View-Controller (MVC). El modelo y el controlador formarán parte del API mientras que la vista será desarrollada de forma totalmente independiente en otro componente. El uso de esta arquitectura busca dividir las responsabilidades del componente en distintas capas, además, cada capa solo podrá interactuar con la inmediatamente superior o inferior.

A continuación detallaremos las capas modelo y controlador propias de este componente.

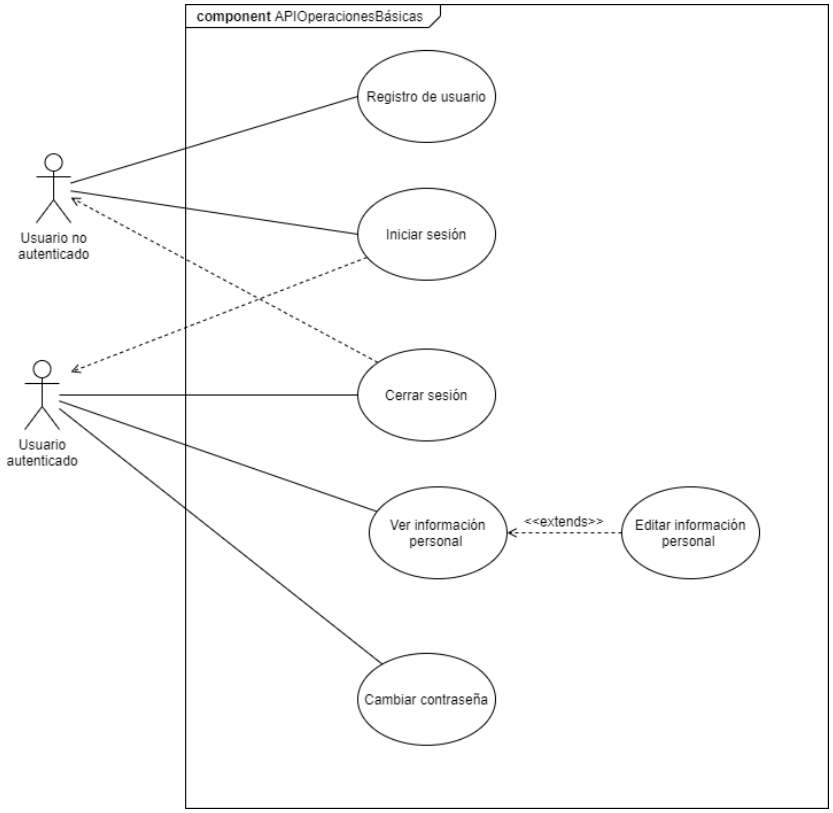

Figura 5.4: Diagrama de Casos de Uso - Gestión de usuarios

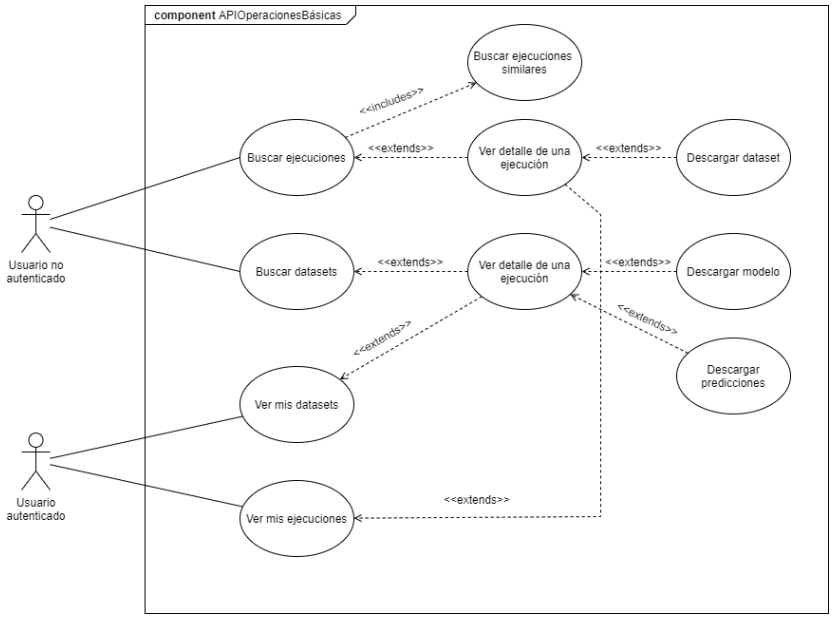

Figura 5.5: Diagrama de Casos de Uso - Gestión de datasets y ejecuciones

Estructura del componente

La distribución de los paquetes que conforman este componente es la siguiente:

- **src/main/java:**
  - **com.cologic.modelab.backend:** Es el paquete principal e incluye la clase que permite arrancar el proyecto Spring Boot y dónde se incluyen algunos *beans* globales.
  - **com.cologic.modelab.backend.common.exceptions:** Este paquete incluye aquellas excepciones definidas para la aplicación y que se usan de forma común en todo el componente.
  - **com.cologic.modelab.backend.model.entities:** Contiene las clases de las entidades persistentes del API. Además de los *DAOs* necesarios para la comunicación con el sistema de persistencia, y, en caso de ser necesario, sus implementaciones.
  - **com.cologic.modelab.backend.model.services:** En este paquete se encuentran las interfaces y las implementaciones de los servicios de la aplicación. Además también encontramos en este paquete aquellas excepciones propias de los servicios.
  - **com.cologic.modelab.backend.rest.common:** Incluye clases comunes a todos los controladores. Se encarga de aspectos como la gestión de excepciones, el control de acceso y la creación de *tokens JWT* para las sesiones.
  - **com.cologic.modelab.backend.rest.controllers:** Este paquete está compuesto por los controladores del API, los cuales exponen los distintos *endpoints* del mismo.
  - **com.cologic.modelab.backend.rest.dtos:** Por último, este paquete incluye los *DTOs* y los conversores de entidades a *DTOs* y viceversa, para el envío de respuestas y la recepción de peticiones.
- **src/main/resources:** Contiene el fichero application.yml que se encarga de la configuración de la aplicación. Además, contiene los ficheros .properties para la internacionalización de mensajes de respuesta.
- **src/test/java:** Contiene las clases en las que se realiza la implementación de las pruebas del API.
- **src/test/resources:** Por último, este paquete contiene aspectos de configuración que solo aplican a los tests.

En la *Figura 5.6* se ilustra la interacción entre los tres paquetes principales del componente (con el fin de mejorar la visualización del mismo se ha decidido no mostrar los atributos de los distintos métodos).

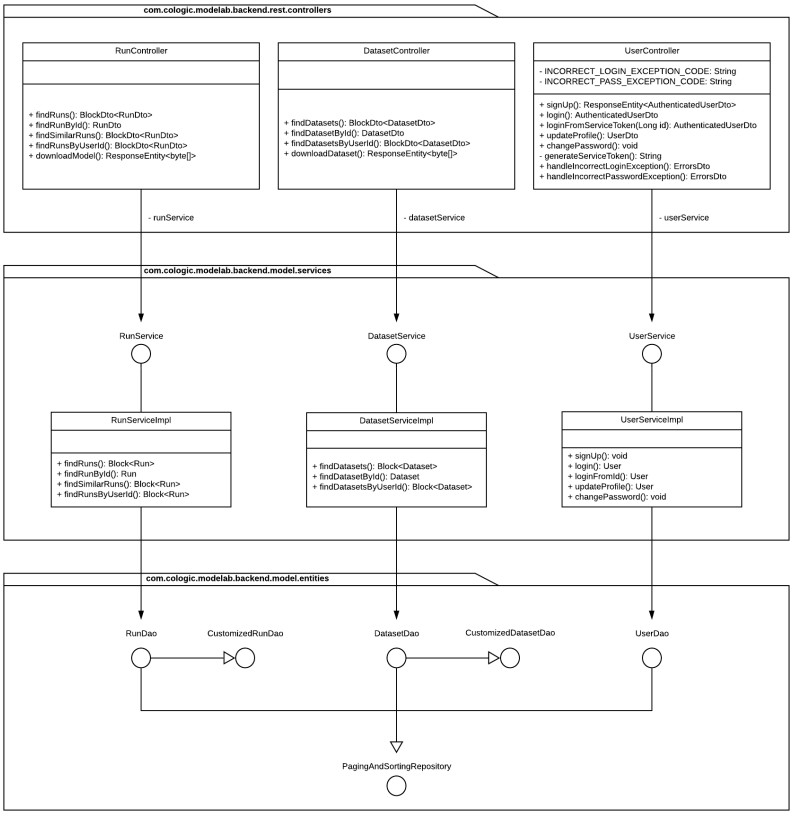

Figura 5.6: Diagrama de clases que muestra la interacción entre los elementos más importantes del API de Opereaciones Básicas

Puntos clave

En este apartado comentaremos los puntos clave en el diseño y desarrollo del componente:

- **Construcción de *Data Access Objects (DAOs)*:** Como podemos observar en la *Figura 5.6* todos los DAO se han desarrollado como interfaces que extienden de *PagingAndSortingRepository*. De esta forma a través del uso de *Spring Data* se podrán construir consultas a través de una convención de nombrado como en el caso del método *findByUserIdOrderByStartMomentDesc* o a través de la anotación *@Query* como se muestra en el fragmento de código 5.3. Además, *Spring Data* ofrece por defecto las operaciones CRUD sin necesidad de implementación para las mismas.

  Por otro lado, en ocasiones puede ser necesario realizar consultas más complejas con casos condicionales. En estos casos, se podrá realizar una implementación personalizada y posteriormente hacer que el DAO extienda de la misma. Como se produce en el fragmento de código 5.3 con *CustomizedProductDao*.

```
1 public interface RunDao extends PagingAndSortingRepository<Run,
      Long>, CustomizedRunDao {
2
3   @Query("SELECT r FROM Run r WHERE (r.dataset.id = ?2 OR r.model =
      ?3) AND r.id != ?1")
4   public Slice<Run> findSimilarRuns(Long runId, Long datasetId,
      String model, Pageable pagelable);
5
6   Slice<Run> findByUserIdOrderByStartMomentDesc(Long userId,
      Pageable pagelable);
7
8 }
```

Fragmento de código 5.3: Repositorio ejecuciones.

- **Servicios:** Para la construcción de los servicios se ha elaborado una interfaz con la firma de los métodos y por otro lado se ha elaborado la implementación de la misma. Esta decisión de diseño busca facilitar el mantenimiento de los servicios y la introducción de nuevas funcionalidades a los mismos. Se ha hecho uso de las anotaciones *@Service* y *@Transactional*, para la creación del *bean* y la gestión de la transaccionalidad respectivamente.

- **Gestión de excepciones:** Para una buena gestión de las excepciones se han creado excepciones comunes a toda la aplicación así como excepciones propias de los servicios. Estas excepciones serán lanzadas desde los servicios y controladas en los controladores mediante los *ExceptionHandlers* proporcionados por *Spring Web*.

  En el fragmento de código 5.4 podemos ver como se realizaría la gestión de una de las excepciones comunes a todos los controladores.

```
1 @ControllerAdvice
2 public class CommonControllerAdvice {
3
4   [...]
5
6   @ExceptionHandler(InstanceNotFoundException.class)
7   @ResponseStatus(HttpStatus.NOT_FOUND)
8   @ResponseBody
9   public ErrorsDto
      handleInstanceNotFoundException(InstanceNotFoundException
      exception, Locale locale) {
10
11    String nameMessage =
      messageSource.getMessage(exception.getName(), null,
      exception.getName(), locale);
```

```
12      String errorMessage =
        messageSource.getMessage(INSTANCE_NOT_FOUND_EXCEPTION_CODE,
13          new Object[] { nameMessage, exception.getKey().toString()
        }, INSTANCE_NOT_FOUND_EXCEPTION_CODE, locale);
14
15      return new ErrorsDto(errorMessage);
16
17    }
18
19    [...]
20
21  }
```

Fragmento de código 5.4: Gestión de excepciones a nivel de controlador.

- **Comentarios:** Otro aspecto fundamental para conseguir un código mantenible y fácilmente comprensible son los comentarios en el mismo. Por ello, para favorecer este aspecto se han establecido una serie de directivas para su uso. En primer lugar, todos los métodos públicos deben contar con JavaDoc[2] explicativo y no simplemente autogenerado. Además, se deberán añadir todos los comentarios necesarios para una fácil compresión del código.

- ***Data Transfer Objects (DTOs)*:** Con el objetivo de ofrecer a los usuarios únicamente la información necesaria y de transformar el cuerpo (JSON) de las peticiones a objetos Java, se han definido los DTOs pertinentes. Un ejemplo sería el implementado en la clase *RunDto* que solo incluye los atributos del objeto *Run* que requiere el API.

- **Controladores:** Por último, para la construcción de los controladores que exponen las operaciones ofrecidas por el API se ha hecho uso del módulo *Spring Web*. Este módulo permite una gestión e implementación de controladores REST de forma muy sencilla y rápida a través de anotaciones.

  Las clases que representan a los controladores se anotarán con *@RestController* y de forma opcional con *RequestMapping('basePath')* para establecer un mapeo base para todos los métodos de la clase. Además de a nivel de clase, podremos anotar los métodos para indicar el *mapping* de los mismos y los parámetros, atributos o cuerpo que formarán parte de las peticiones que lleguen a dicho *endpoint*.

  Como añadido a todo esto, la conversión del cuerpo de las peticiones (JSON) a los objetos Java (DTOs) se realiza de forma totalmente automática.

  En el fragmento de código 5.5 podemos ver un ejemplo de lo comentado anteriormente.

[2] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

```
@RestController
@RequestMapping("/users")
public class UserController {

  [...]

  @PutMapping("/{id}")
  public UserDto updateProfile(@RequestAttribute Long userId,
    @PathVariable("id") Long id,
      @Validated({ UserDto.UpdateValidations.class }) @RequestBody
    UserDto userDto)
      throws InstanceNotFoundException, PermissionException {

    if (!id.equals(userId)) {
      throw new PermissionException();
    }

    return toUserDto(
        userService.updateProfile(id, userDto.getFirstName(),
    userDto.getLastName(), userDto.getEmail()));

  }

  [...]

}
```

Fragmento de código 5.5: Controlador para la gestión de usuarios.

5.5 Aplicación frontend

En este apartado comentaremos de forma detallada diferentes aspectos relacionados con la implementación de la aplicación frontal que consumirán los usuarios para acceder a los dos APIs comentadas anteriormente.

Además mostraremos también en este apartado la apariencia final de la plataforma desde el punto de vista de un usuario.

5.5.1 Diseño e implementación

En este apartado comentaremos las decisiones de diseño e implementación que se han seguido para el desarrollo de la aplicación frontal de la plataforma.

Estructura del componente

La estructura que sigue la aplicación frontal es la siguiente:

- **src/backend/:** Contiene la implementación de las llamadas a los APIs.
- **src/modules/:** Este directorio contiene la implementación de todos los componentes de la aplicación, organizados a su vez en subdirectorios.
- **src/modules/store/:** En este directorio están los ficheros que realizan la configuración del estado global de la aplicación.
- **src/i18n/:** Está formado por los ficheros que contienen los textos en diferentes idiomas para la internacionalización de la aplicación.

Puntos clave

Aquí se detallan los puntos claves en el diseño y el desarrollo de la aplicación frontal de la plataforma:

- **Comunicación con los APIs:** Para realizar las peticiones desde el frontal a los APIs de la plataforma se ha hecho uso del API Fetch. Este API permitirá realizar consultas al *backend* de la plataforma de forma sencilla y potente. Además, ofrece una funcionalidad extremadamente importante para la plataforma, permite realizar dichas peticiones de forma totalmente asíncrona, ofreciendo así al usuario final una mejor experiencia de uso.
- **Diseño basado en componentes:** Mediante el *framework* ReactJs se crean componentes encapsulados que manejen su propio estado, y conviértelos en interfaces de usuario complejas. Ya que la lógica de los componentes está escrita en JavaScript y no en plantillas, puedes pasar datos de forma sencilla a través de tu aplicación y mantener el estado fuera del DOM.

  En el fragmento de código 5.6 podemos ver un ejemplo de un componente sencillo y totalmente reutilizable que se encarga de la paginación de cualquier búsqueda dentro de la plataforma.

```
const Pager = ({ back, next }) => (
  <nav aria-label="page navigation">
    <ul className="pagination justify-content-center">
      <li className={`page-item ${back.enabled ? "" : "disabled"}`}>
        <button className="page-link" onClick={back.onClick}>
          <FormattedMessage id="project.global.buttons.back" />
        </button>
```

```
      </li>
      <li className={`page-item ${next.enabled ? "" : "disabled"}`}>
        <button className="page-link" onClick={next.onClick}>
          <FormattedMessage id="project.global.buttons.next" />
        </button>
      </li>
    </ul>
  </nav>
);
```

Fragmento de código 5.6: Componente Pager para la paginación en frontal.

- **Gestión del estado de los componentes:** Como hemos indicado en el punto anterior React permite gestionar el estado de los componentes dentro de los mismos de tal forma que este podrá cambiar produciendo así un nuevo renderizado, únicamente de ese componente.

  La gestión de estados en la mayoría de componentes se ha realizado mediante el uso del *hook useState*, mediante el cual definimos un estado inicial y una serie de funciones que lo pueden cambiar.

  En el fragmento de código 5.7 se muestra un ejemplo del manejo del estado de un componente. El fragmento muestra como mediante el estado del componente se gestiona el cambio entre las pestañas que ofrece el mismo.

```
const [tab, setTab] = useState("overview");

const handleTabsChange = (event, newValue) => {
  setTab(newValue);
};
```

Fragmento de código 5.7: Manejo del estado de un componente mediante el uso del *hook useState*.

- **Gestión del estado entre componentes:** Además de la existencia de un estado propio de cada componente se ha hecho uso de la librería Redux para permitir la existencia de un estado compartido entre los distintos componentes, permitiendo así una mejor fragmentación en los mismos. De igual forma que en el punto anterior, se ha hecho uso de los *hooks useDispatch* y *useSelector* para la gestión de estos estados.

- **Diseño *responsive*:** Por último, se ha hecho uso de la librería Material UI para diseñar e implementar un frontal totalmente *responsive*, permitiendo así, el uso de la plataforma desde todo tipo de dispositivos. Esta librería, adaptada a su uso en React, ofrece una

serie de componentes por defecto que permiten un desarrollo ágil y con un resultado estético muy bueno.

Algunos ejemplos del diseño *responsive* del sistema se mostrarán a continuación en la sección en la que se muestra el resultado final de la interfaz web de la misma.

5.5.2 Interfaz de la plataforma

En esta sección se presentará el resultado final de la interfaz de la plataforma.

En primer lugar, en las *Figuras 5.7 y 5.8* se muestra la página de inicio de la plataforma.

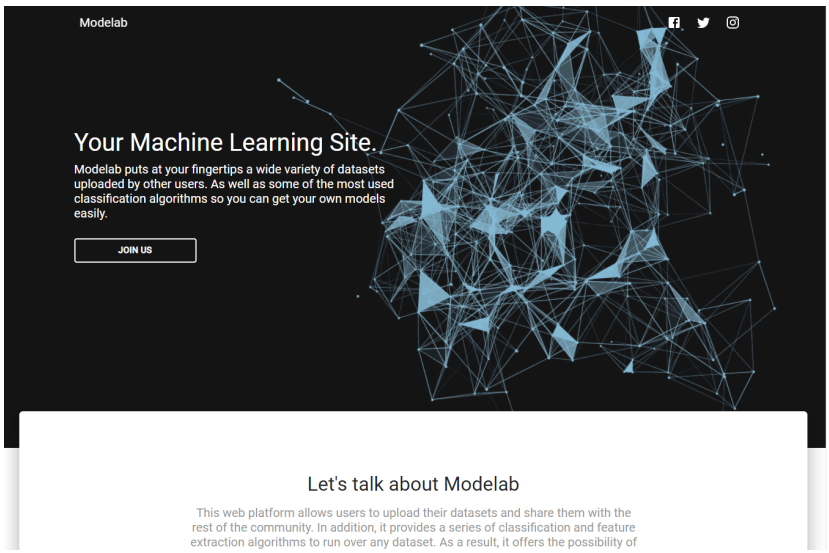

Figura 5.7: Interfaz web - Página de inicio.

En las *Figuras 5.9 y 5.10* se muestran las páginas de registro de usuarios e inicio de sesión.

Los usuarios tendrán acceso a su perfil donde podrán ver los *datasets* que han subido y la ejecuciones que han realizado, como podemos ver en la *Figura 5.11*. Además, podrán editar su información personal y cambiar la contraseña de acceso (*Figuras 5.12 y 5.13*).

Por otro lado, los usuarios podrán subir *datasets* a la plataforma y realizar búsquedas sobre ellos, pudiendo ver los detalles y una vista previa de los mismos junto con un análisis de sus *features* (*Figuras 5.14, 5.15, 5.16 y 5.17*). Además, desde la vista de detalles de un *dataset* podremos proceder a su descarga o comenzar una clasificación sobre él, a través del formulario representado en la *Figura 5.18*.

Como se representa en las *Figuras 5.19 y 5.20* los usuarios podrán consultar las ejecuciones realizadas en la plataforma, y los detalles de las mismas. Desde la página de detalles podrán descargar el modelo resultante del entrenamiento y las predicciones realizadas.

Como cierre para esta sección mostraremos algunos ejemplos del diseño *responsive* del frontal (*Figuras 5.21 y 5.22*).

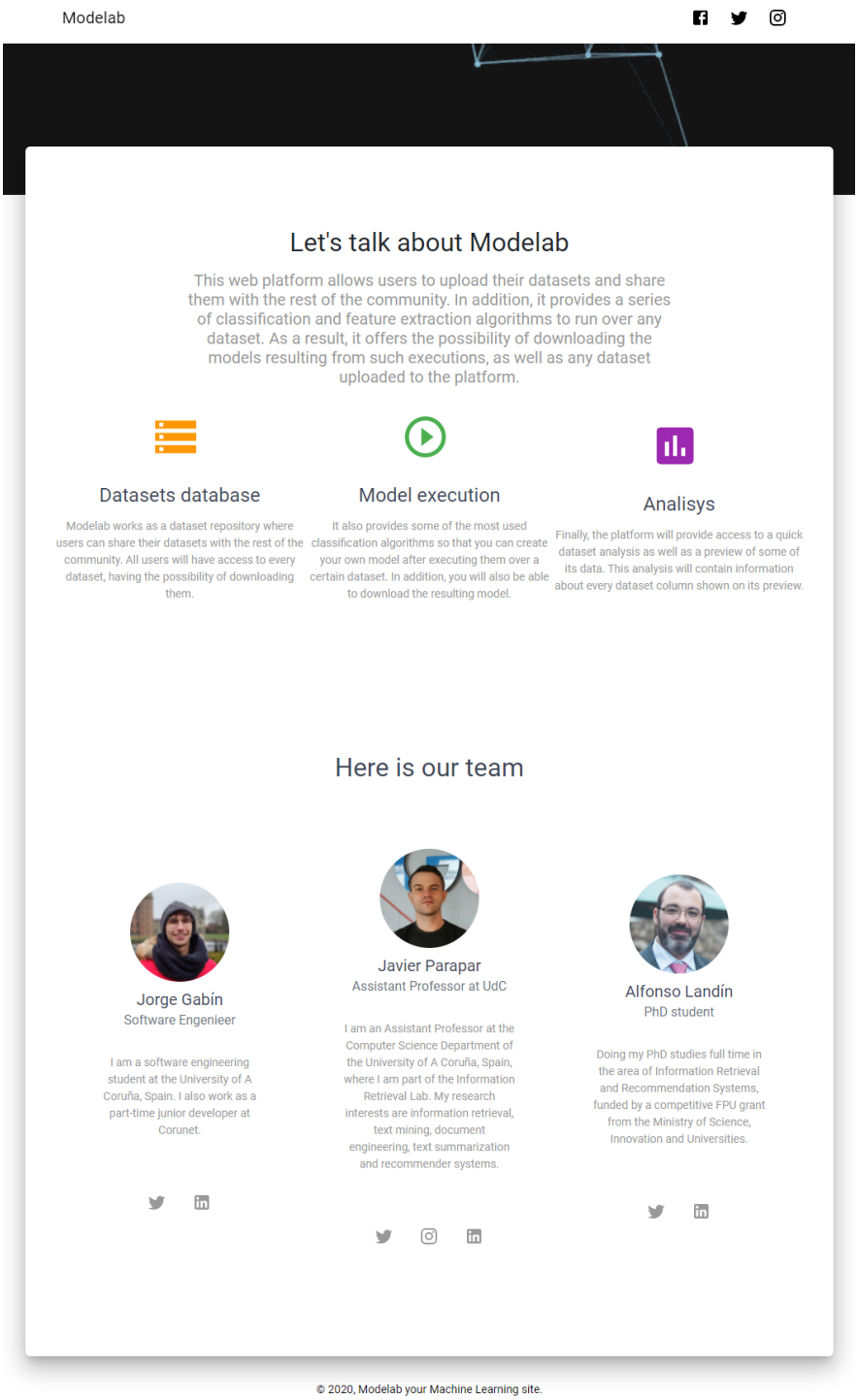

Figura 5.8: Interfaz web - Página de inicio.

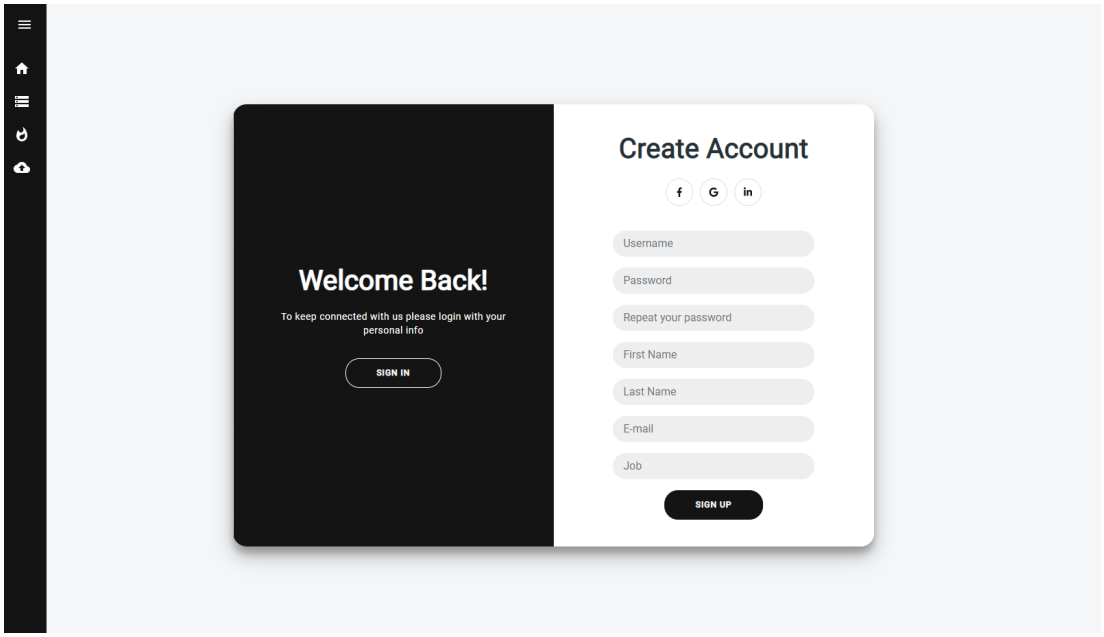

Figura 5.9: Interfaz web - Registro.

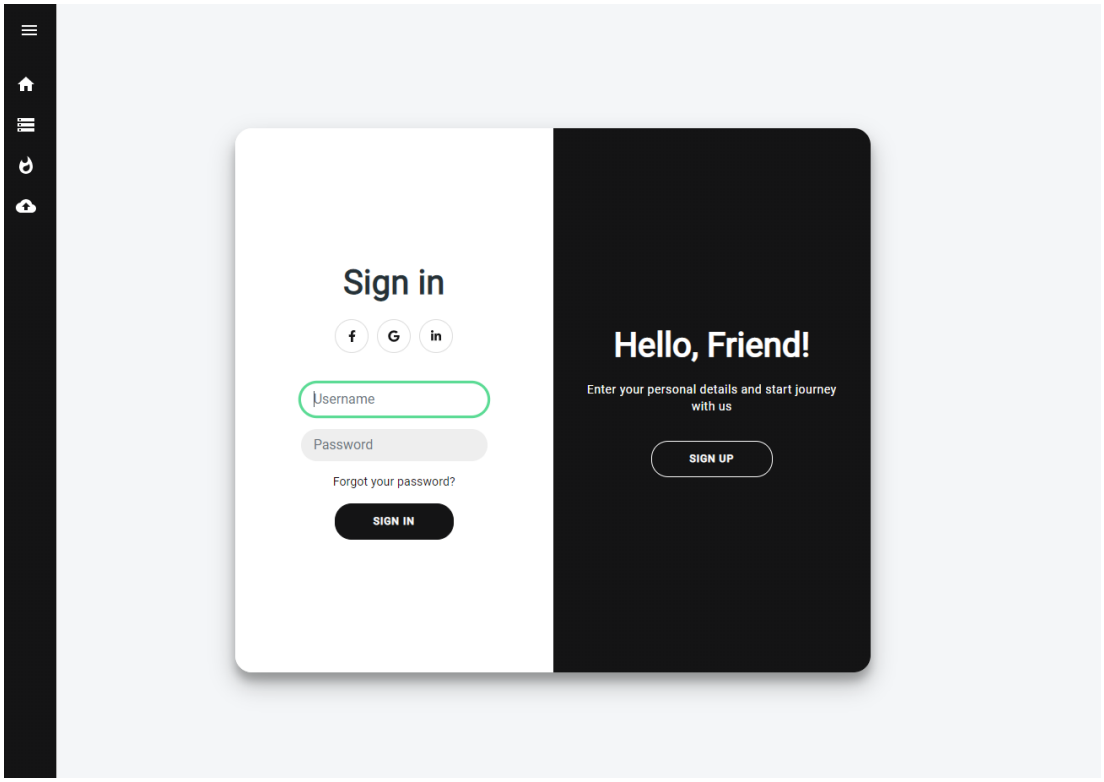

Figura 5.10: Interfaz web - Inicio de sesión.

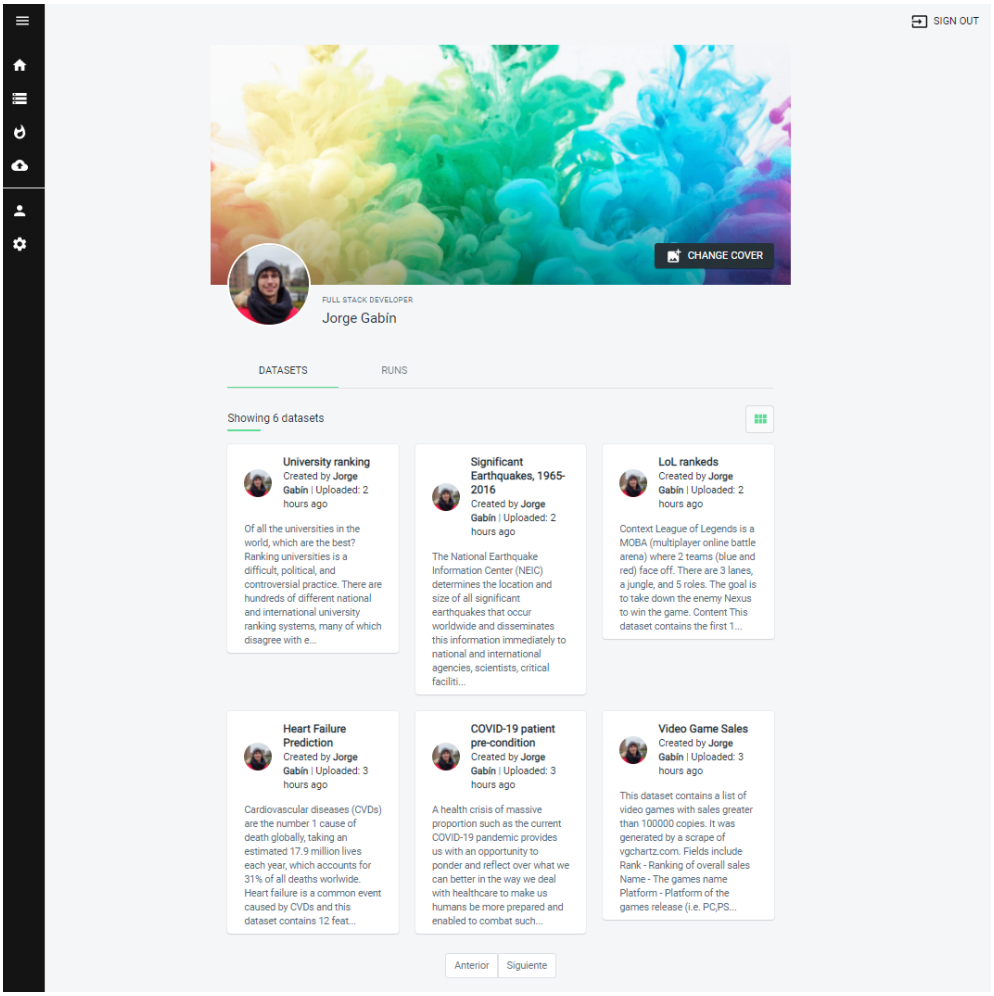

Figura 5.11: Interfaz web - Perfil de un usuario.

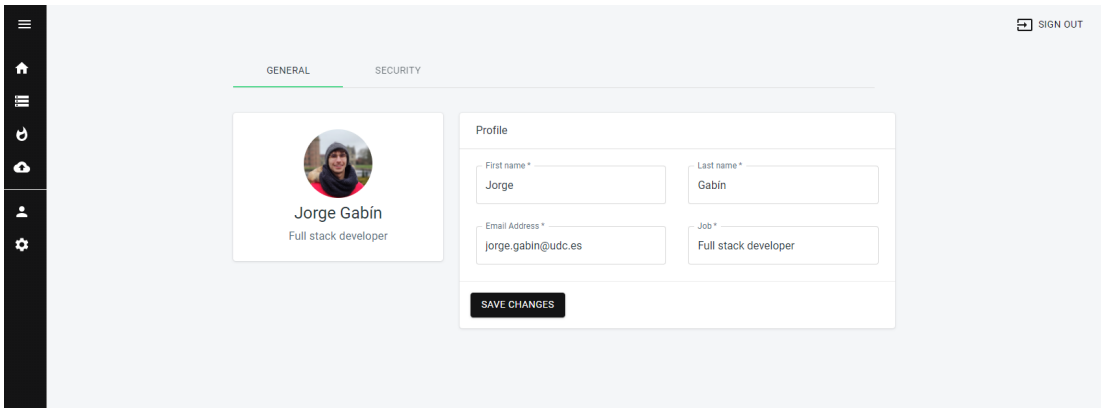

Figura 5.12: Interfaz web - Editar información de un usuario.

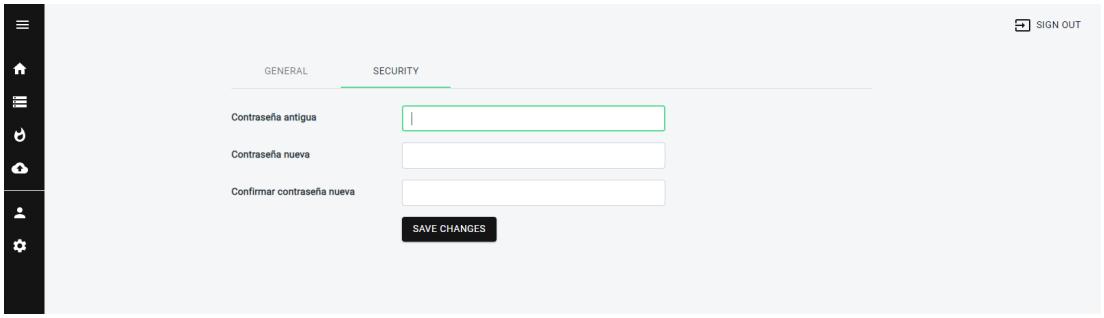

Figura 5.13: Interfaz web - Cambiar contraseña.

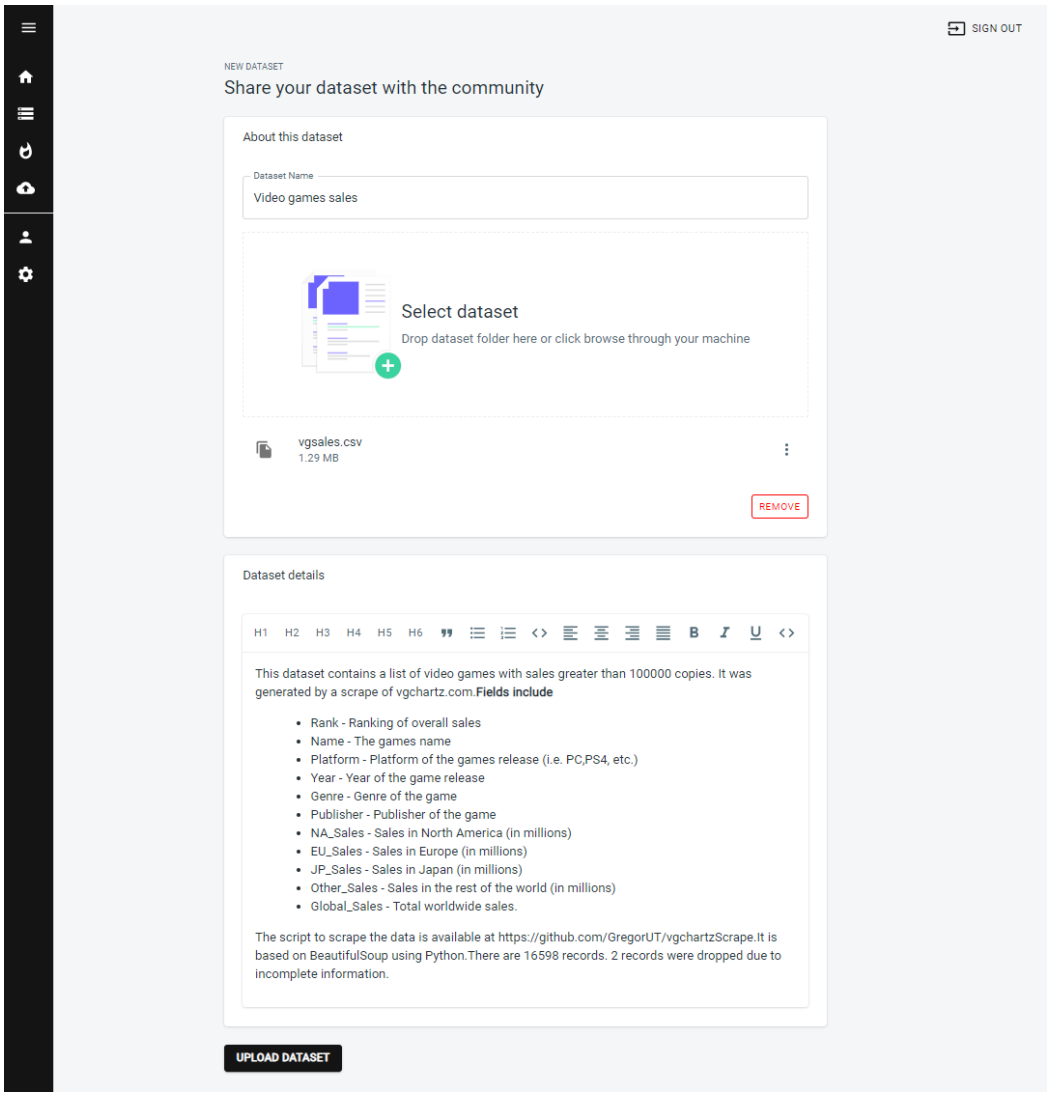

Figura 5.14: Interfaz web - Subida de datasets.

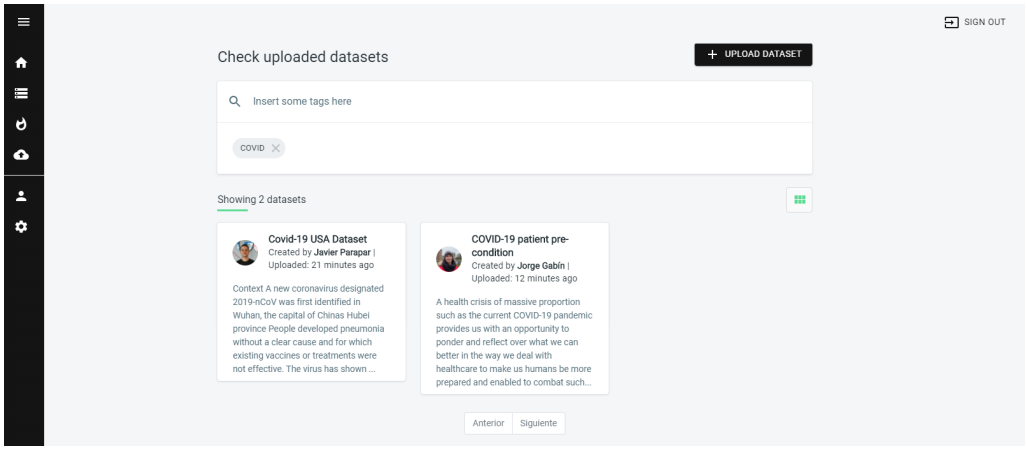

Figura 5.15: Interfaz web - Búsqueda de datasets con filtro.

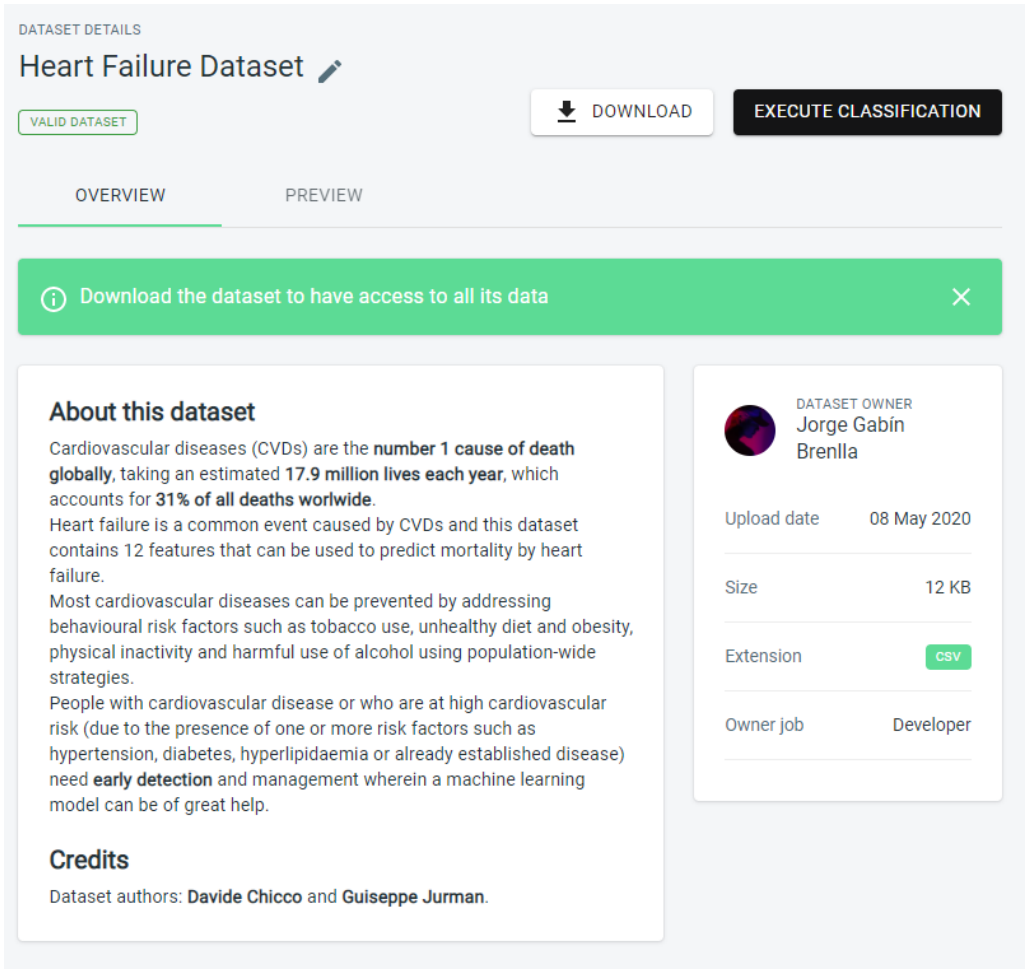

Figura 5.16: Interfaz web - Detalles dataset.

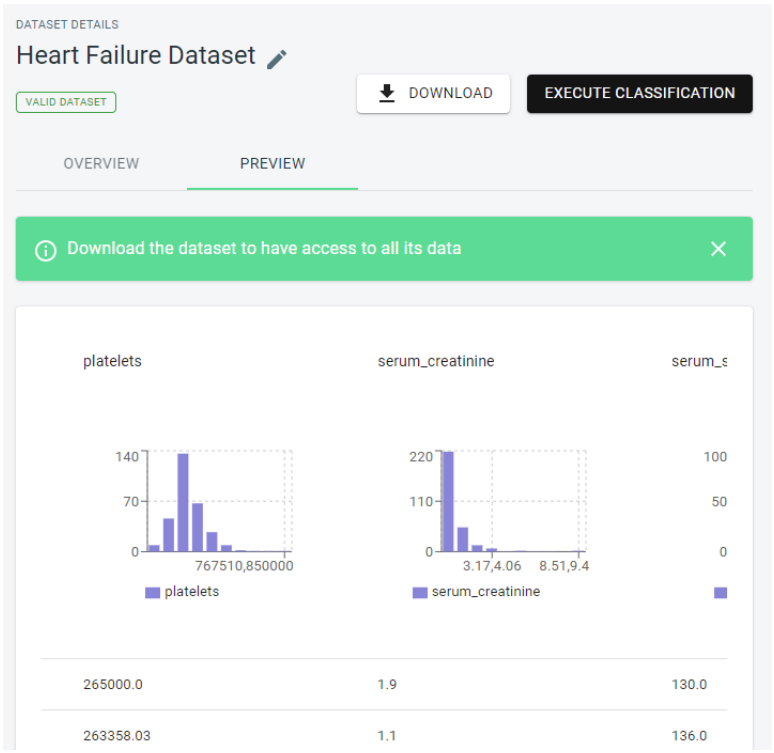

Figura 5.17: Interfaz web - Detalles dataset (vista previa).

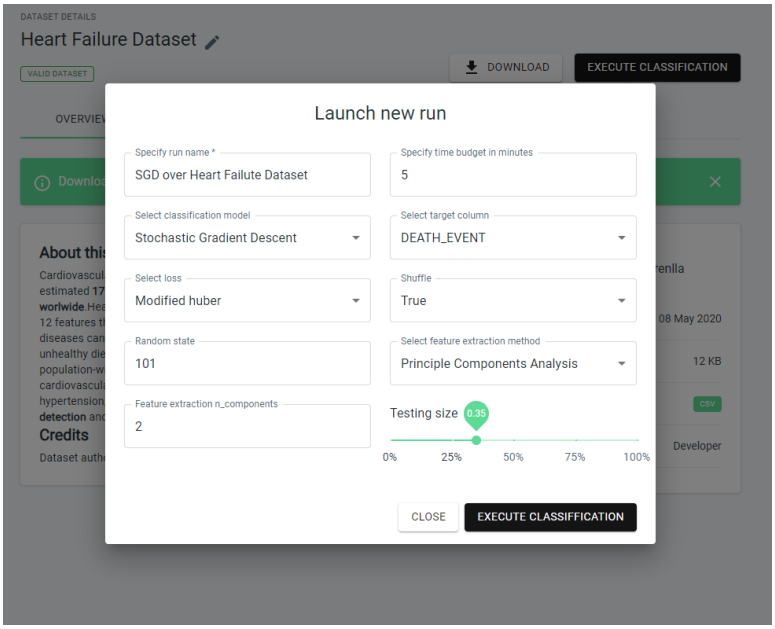

Figura 5.18: Interfaz web - Modal para la ejecución de una clasificación.

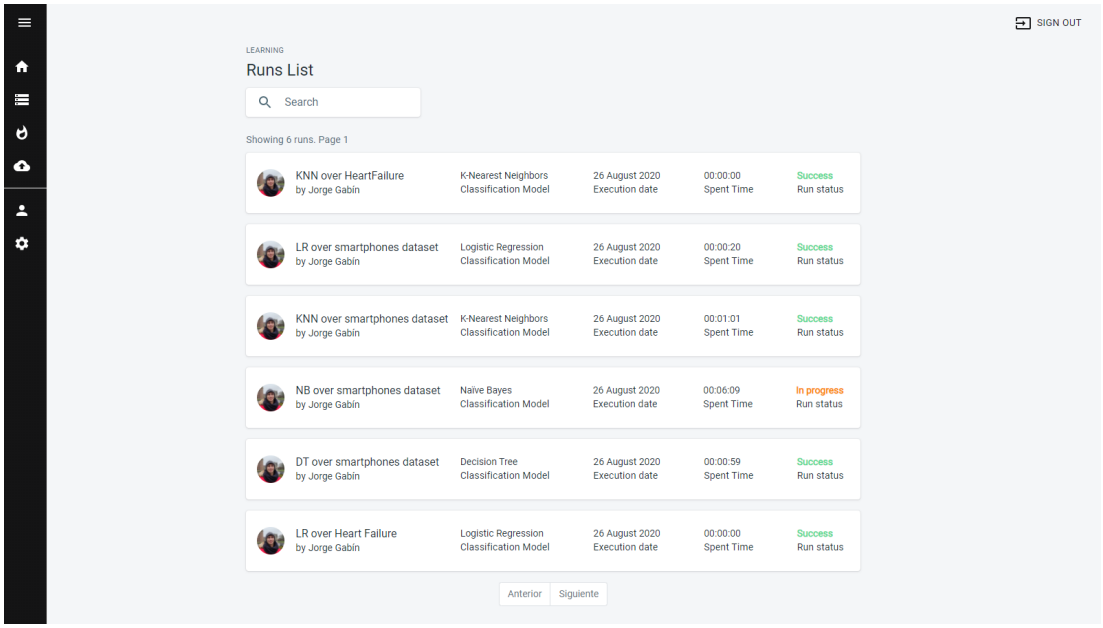

Figura 5.19: Interfaz web - Lista de ejecuciones.

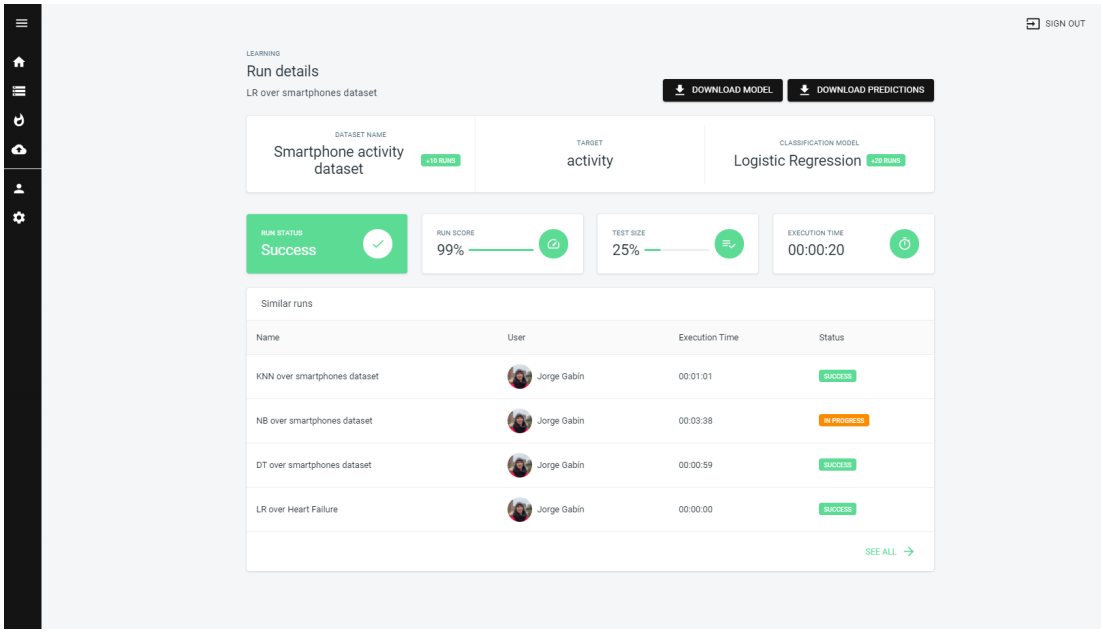

Figura 5.20: Interfaz web - Detalle ejecuciones.

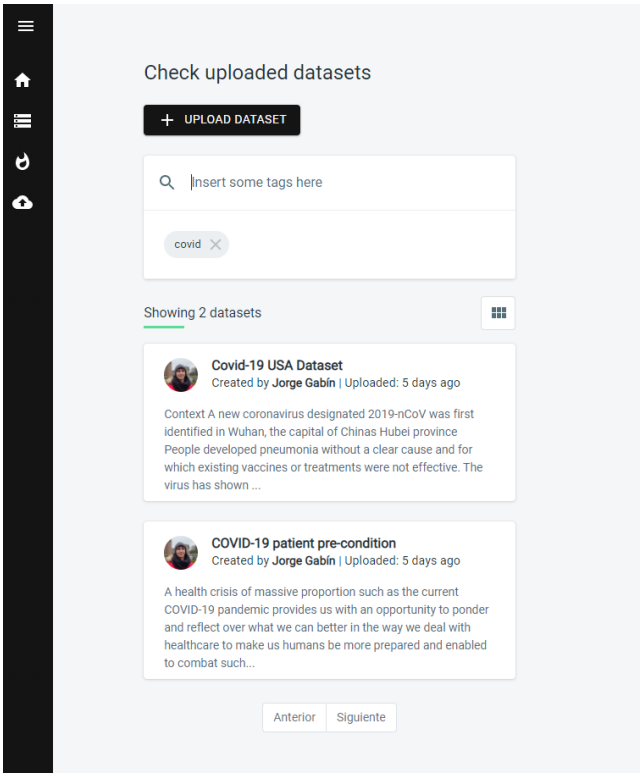

Figura 5.21: Interfaz responsive - Búsqueda de datasets con filtro.

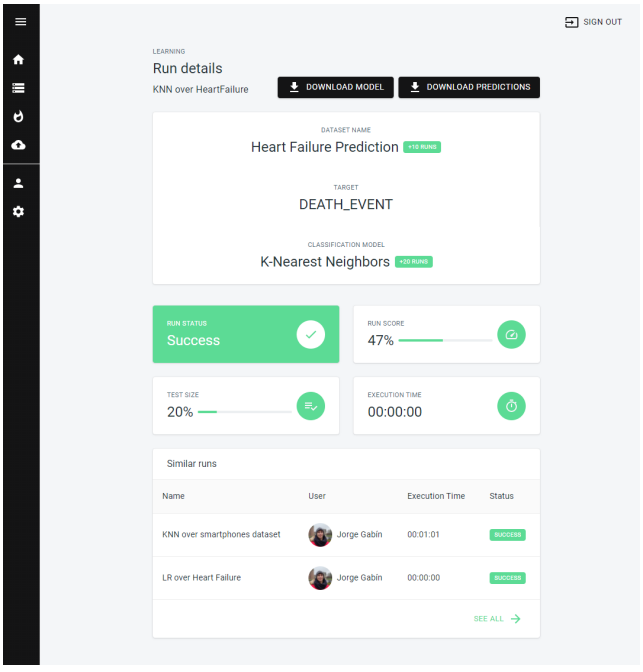

Figura 5.22: Interfaz responsive - Detalle ejecuciones.

5.6 Pruebas

En un proyecto software es imprescindible contar con una *suite* de pruebas completa y que asegure el buen funcionamiento del mismo. Por ello, en este apartado haremos hincapié en la *suite* de pruebas desarrollada para el API de operaciones básicas. Para dicho componente se han empleado todo tipo de pruebas: unidad, integración y aceptación; las cuales comentaremos en detalle a continuación. Por otro lado, debido al carácter del proyecto encargado de los aspectos relacionados con la ejecución de clasificaciones y el análisis de *datasets* se han realizado únicamente pruebas de integración y aceptación de forma manual.

5.6.1 Pruebas de unidad

Mediante las pruebas de unidad se trata de probar el correcto funcionamiento de un componente concreto de forma aislada. Por ejemplo, la implementación de un servicio o un controlador *mockeando* las relaciones con el resto de componentes. Como se ha comentado en el Capítulo 3 de la memoria, se ha hecho uso del *framework* Mockito para el desarrollo de las mismas.

5.6.2 Pruebas de integración

Los *tests* de integración permiten comprobar el funcionamiento de los componentes del sistema de forma conjunta. Para ello, se han implementado pruebas a nivel de proyecto usando el *framework* Spring-boot Test y, por otro lado, se han probado ambos APIs mediante la realización de peticiones desde un *API tester*. El utilizado en el presente proyecto ha sido Postman[3].

5.6.3 Pruebas de aceptación

Por último, y a través del frontal de la plataforma se han realizado las pruebas de aceptación pertinentes. Además, de haber sido realizadas por el propio alumno, personas ajenas al proyecto han participado en estas pruebas.

5.7 Herramientas de ayuda al desarrollo y monitorización del sistema

Además del desarrollo de la plataforma, con el fin de cumplir con los requisitos de calidad y disponibilidad establecidos se ha hecho uso de distintas herramientas de monitorización e integración e inspección continua.

[3] https://www.getpostman.com

5.7.1 Integración continua

Para llevar a cabo la integración continua del proyecto se ha configurado Jenkins. Mediante el uso de esta herramienta se ejecuta una construcción del proyecto Java ante cada actualización del repositorio, asegurando así la correcta ejecución de los tests.

En la *Figura 5.23* podemos apreciar un resumen del histórico de las *builds* del proyecto, en las que podemos apreciar la evolución de los tests.

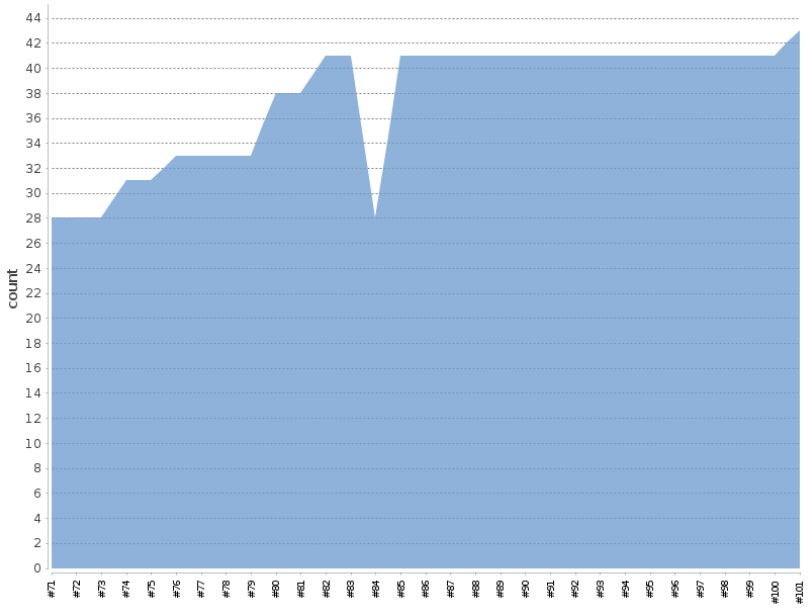

Figura 5.23: Histórico de las últimas *builds* del proyecto con la evolución de las pruebas en el mismo.

5.7.2 Inspección continua

Por otro lado, con el fin de evaluar la calidad del software desarrollado se han empleado dos *linters*: SonarQube y ESLint. El primero se ha configurado de tal forma que se realice una ejecución que evalúe la calidad del código Java del repositorio por cada ejecución de Jenkins. Por otro lado, ESLint se ha ejecutado de forma local para evaluar la calidad del código JavaScript del frontal. Esta herramienta se ha configurado para que también se ejecute cada vez que se compile el proyecto en local.

En la *Figura 5.24* podemos ver algunos parámetros de calidad obtenidos mediante SonarQube.

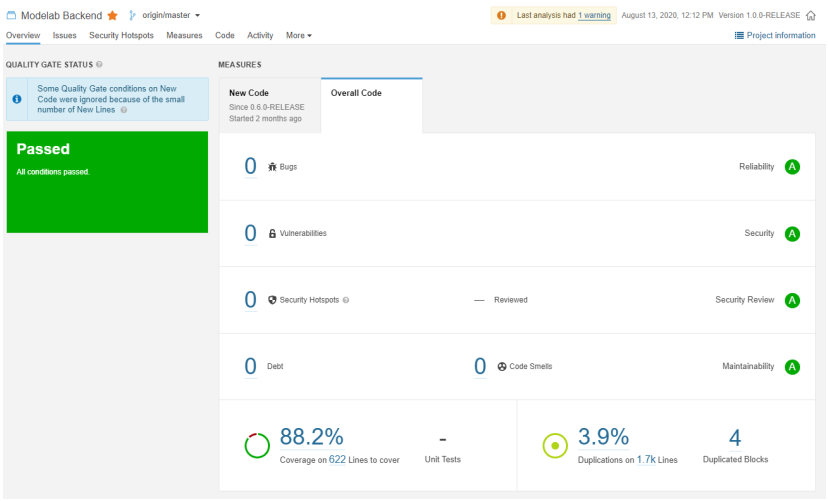

Figura 5.24: Resultado de las métricas de Sonar al finalizar el proyecto.

5.7.3 Monitorización

Además, de la gestión de la calidad del código se ha configurado un sistema de logs y otro de métricas para poder llevar a cabo una buena monitorización del sistema.

Ambos sistemas han sido desplegados en Docker, mediante Docker Compose, junto con las bases de datos necesarias para cada uno, así como el motor de búsqueda necesario para el sistema de logs.

Graylog

La herramienta de logs que se ha utilizado ha sido Graylog, la cual se apoyará en una base de datos documental, MongoDB; y un motor de búsqueda, Elasticeearch, para el manejo y gestión de los mismos.

Graylog ofrecerá por lo tanto una interfaz web a través de la cual podremos consultar los logs que hayamos registrado usando distintos criterios de búsqueda y con una gran velocidad.

En la *Figura 5.25* se muestra una búsqueda de los logs de la aplicación.

Grafana

Por otro lado, la herramienta escogida para la monitorización de distintas métricas del sistema ha sido Grafana. Como ya hemos dicho esta herramienta también ha sido desplegada en un Docker junto con la base de datos InfluxDB en la cual se guardarán los datos de las métricas que consultará Grafana para representarlos en sus respectivos paneles.

En la *Figura 5.26* podemos ver algunas de las métricas que se han representado mediante esta herramienta.

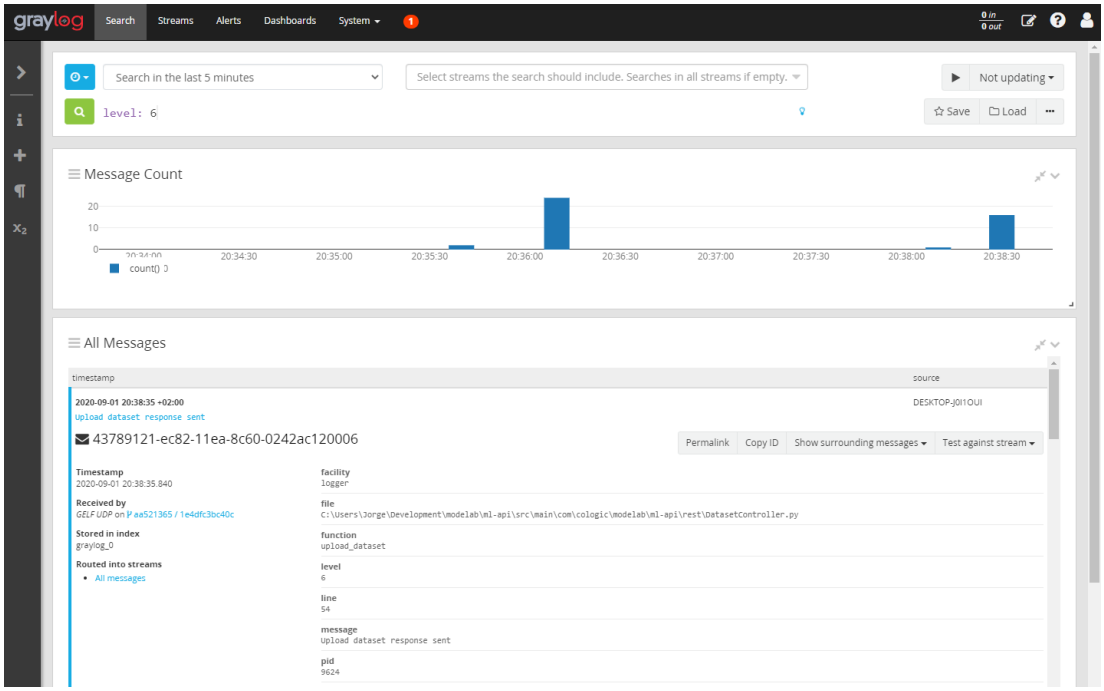

Figura 5.25: Ejemplo de un log almacenado en Graylog.

Figura 5.26: Captura de alguna de las métricas mostradas en Grafana.

Capítulo 6

Conclusiones

El enorme crecimiento del volumen de datos almacenados en la nube, junto con la necesidad de analizar dichos datos, ha provocado el nacimiento de nuevas tecnologías y herramientas que permitan llevar a cabo estos análisis de forma sencilla sin necesidad de ser expertos en la materia. En este contexto es donde se encuadra la plataforma descrita a lo largo de este documento, como una herramienta que permite la introducción de nuevos usuarios al mundo del *machine learning*, y más concretamente a la clasificación de datos; pero también, como una herramienta que permite agilizar sus procesos a aquellos más experimentados.

6.1 Características del proyecto

Tras varios meses de desarrollo, se ha conseguido desarrollar una plataforma web que cumple con los objetivos establecidos en un comienzo, es decir, una plataforma en la cual los usuarios podrán almacenar y acceder a *datasets* sobre los cuales podrán ejecutar una serie de algoritmos de clasificación y consultar los resultados de los mismos.

A continuación se enumeran las principales características de la plataforma:

- Sistemas de almacenamiento independientes según la naturaleza de los datos.
- Componentes independientes y separados por responsabilidades para una mayor escalabilidad.
- Funcionamiento asíncrono de operaciones pesadas como la subida de *datasets* o la ejecución de clasificaciones para una mejor experiencia de usuario.
- Interfaz de usuario *responsive* (apta para su uso desde móviles y tablets) y pensada para obtener la mejor experiencia de usuario posible.
- Sistema de monitorización eficiente y escalable, para asegurar un buen funcionamiento del sistema.

6.2 Relación con las competencias del grado

Durante el desarrollo del proyecto se ha hecho uso de una gran cantidad de los conocimientos adquiridos a lo largo del grado. A continuación se resumen aquellas competencias que se han aplicado a lo largo del desarrollo.

En primer lugar, tanto en el análisis de las posibles metodologías como en la ejecución de la escogida (Scrum) los conocimientos adquiridos en la asignatura *Metodologías de Desarrollo* han sido fundamentales para la toma de toma de decisiones tras el análisis y para una buena ejecución de la misma.

Por otro lado, asignaturas como *Ingeniería de Requisitos*, *Diseño Software* y *Arquitectura Software* han sido de gran ayuda para la elección de la arquitectura del proyecto y la correcta implementación de la misma.

También me han resultado de gran utilidad los conocimientos adquiridos en las asignaturas de *Bases de Datos* y *Bases de Datos Avanzadas* para la elaboración de un buen modelo de datos y la elaboración de consultas rápidas y eficientes.

Para una buena gestión y planificación del proyecto y para obtener la máxima calidad posible en el mismo han sido de gran utilidad los conocimientos adquiridos en *Gestión de Proyectos*, *Proyectos de Desarrollo Software* y *Aseguramiento de la Calidad*.

En la elaboración del API de Operaciones Básicas y del frontal de la plataforma, se han aplicado los conocimientos adquiridos en las asignaturas de *Internet y Sistemas Distribuidos* y *Programación Avanzada*. Además para conseguir una buena experiencia de usuario se han seguido las guías estudiadas en la asignatura *Interfaces Persona Máquina*. Además la introducción al lenguaje de programación Python que se realiza en la misma ha sido un buen comienzo para el desarrollo del API de*Machine Learning*.

Las competencias adquiridas en la asignatura de *Sistemas Inteligentes* han sido de gran ayuda como introducción al mundo del aprendizaje automático en el cual se ha profundizado bastante a lo largo del desarrollo de este proyecto.

Además, los conocimientos adquiridos en *Herramientas de Desarrollo*, donde se explica la necesidad de seguir unas buenas prácticas y utilizar de forma adecuada herramientas de gestión y automatización (Git, Eclipse, Maven, Jenkins, Sonar...), me han ayudado a realizar un desarrollo exitoso y a saber utilizar de forma eficiente las distintas herramientas para la gestión de proyectos y software utilizadas.

Por último, las Prácticas en Empresa me han aportado muchos conocimientos tanto a nivel tecnológico, como a nivel de metodologías de trabajo que he podido aplicar para el desarrollo de este proyecto.

6.3 Trabajo futuro

Tras finalizar el proyecto se plantean distintos aspectos y nuevas vías de trabajo para mejorar y ampliar sus funcionalidades, y para su explotación comercial:

- Implementación de nuevos algoritmos, ampliar la oferta en el campo de la clasificación y comenzar a trabajar en otros campos como por ejemplo el clustering.
- Permitir el uso de Python Notebooks para que los usuarios puedan subir sus propios algoritmos.
- Migrar el almacenamiento de *datasets* a la nube.
- Explotación comercial de la plataforma.

Apéndices

Lista de acrónimos

API Application Programming Interface. iii, v, 2, 13, 14, 16, 34, 35, 40–44, 46, 47, 49, 50, 52–54, 65, 70

DAOs Data Access Objects. 50

DTOs Data Transfer Objects. 52

IDC International Data Corporation. 1

IDE Integrated Development System. 20

K-NN K-Nearest Neighbors. 5, 7

MVC Model-View-Controller. 47

SGD Stochastic Gradient Descent. 8

SVM Support Vector Machine. 11

Glosario

clasificador Algoritmo que, recibiendo como entrada cierta información de un objeto, es capaz indicar la categoría o clase a que pertenece de entre un número acotado de clases posibles. 7

Inteligencia Artificial Es la inteligencia llevada a cabo por máquinas. En ciencias de la computación, una máquina «inteligente» ideal es un agente flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea. 5

inteligencia de negocio Conjunto de procesos, aplicaciones y tecnologías que facilitan la obtención rápida y sencilla de datos provenientes de los sistemas de gestión empresarial para su análisis e interpretación, de manera que puedan ser aprovechados para la toma de decisiones y se conviertan en conocimiento para los responsables del negocio. 1

machine learning Disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente a partir de conjuntos de datos. 2, 3, 14, 69

modelo Un modelo en el campo del *machine learning* es una representación matemática de un proceso del mundo real. 6, 8, 10, 34, 35

zettabytes Unidad de almacenamiento de información cuyo símbolo es el ZB, equivale a 1021 bytes. 1

Bibliografía

[1] C. C. Aggarwal, *Data Classification: Algorithms and Applications*. Chapman and Hall/CRC, 2014.

[2] A. Geron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly UK Ltd., 2019.

[3] S. L. David W. Hosmer, *Applied Logistic Regression*. Wiley-Interscience Publication, 2000.

[4] O. Kramer, *K-Nearest Neighbors*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 13–23. [En linea]. Disponible en: https://doi.org/10.1007/978-3-642-38652-7_2

[5] L. M. Surhone, *Naive Bayes Classifier*. Betascript Publishers, 2010.

[6] L. Bottou, *Stochastic Gradient Descent Tricks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. [En linea]. Disponible en: https://doi.org/10.1007/978-3-642-35289-8_25

[7] C. Smith, *Decision Trees and Random Forests: A Visual Introduction For Beginners*, 2017.

[8] Y. Liu, Y. Wang, and J. Zhang, "New machine learning algorithm: Random forest," in *Information Computing and Applications*, B. Liu, M. Ma, and J. Chang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 246–252. [En linea]. Disponible en: https://doi.org/10.1007/978-3-642-34062-8_32

[9] A. C. Ingo Steinwart, *Support Vector Machines*. Springer, 2008.

[10] M. N. Isabelle Guyon, Steve Gunn and L. A. Zadeh, *Feature Extraction: Foundations and Applications*. Springer, 2006.

[11] D. Beazley and B. K. Jones, *Python Cookbook*. O'Reilly Media, Inc., 2013.

[12] G. M. Raul Garreta, *Learning scikit-learn: Machine Learning in Python.* Packt Publishing, 2013.

[13] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.* O'Reilly Media, 2012.

[14] G. Dwyer, S. Aggarwal, and J. Stouffer, *Flask: Building Python Web Services.* Packt Publishing, 2017.

[15] H. Schildt, *Java: The Complete Reference, Eleventh Edition.* McGraw-Hill Education, 2018.

[16] C. Walls, *Spring in Action.* Manning Publications, 2014.

[17] ——, *Spring Boot in Action.* Manning Publications, 2015.

[18] M. Grzejszczak, *Mockito Cookbook.* Packt Publishing, 2014.

[19] R. S. Shekhar Gulati, *Java Unit Testing with JUnit 5: Test Driven Development with JUnit 5.* Apress, 2017.

[20] A. Freeman, *Pro React 16.* Apress, 2019.

[21] A. Boduch, *React Material-UI Cookbook: Build captivating user experiences using React and Material-UI.* Packt Publishing, 2019.

[22] K. Schwaber, "Scrum development process," in *Business Object Design and Implementation*, J. Sutherland, C. Casanave, J. Miller, P. Patel, and G. Hollowell, Eds. London: Springer London, 1997, pp. 117–134. [En linea]. Disponible en: https://doi.org/10.1007/978-1-4471-0947-1_11